

EXHIBIT E

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

GOOGLE LLC,
Petitioner,

v.

SINGULAR COMPUTING LLC,
Patent Owner.

Case No. TBD
Patent No. 8,407,273

DECLARATION OF RICHARD GOODIN

Google Exhibit 1003 Google v. Singular

CONDENSED TABLE OF CONTENTS

I.	PERSONAL AND PROFESSIONAL BACKGROUND	1
II.	MATERIALS REVIEWED AND CONSIDERED	3
III.	MY UNDERSTANDING OF PATENT LAW	5
	A. Anticipation	7
	B. Obviousness.....	7
IV.	THE '273 PATENT	10
	A. Overview	10
	B. Claims	20
	C. Person of Ordinary Skill in the Art	21
	D. Prosecution History	24
V.	BATES-2010 RENDERS OBVIOUS CLAIMS 1-70.....	24
	A. Bates-2010 Is Prior Art Because the Challenged Claims Are Not Entitled to the '201 Application's Priority Date	24
	B. Written Description: The '201 Application Does Not Demonstrate Possession of the Full Scope of Any of the Challenged Claims	25
	C. Enablement: The '201 Application Does Not Enable the Full Scope of Any Challenged Claim.....	69
	D. Bates-2010 Renders Obvious Claims 1-70	74
VI.	CLAIMS 1-2, 21-24, 26, AND 28 WOULD HAVE BEEN OBVIOUS OVER DOCKSER.....	122
	A. Dockser (Ex. 1007).....	123
	B. Claim 1	131
	C. Claim 2: "The device of claim 1, wherein the at least one first LPHDR execution unit comprises at least part of an FPGA"	209
	D. Claim 21: "The device of claim 1, wherein the device includes memory locally accessible to the at least one first LPHDR execution unit."	209
	E. Claim 22: "The device of claim 1, wherein the device is implemented on a silicon chip"	212

F. Claim 23: “The device of claim 1, wherein the device is implemented on a silicon chip using digital technology.”	213
G. Claim 24: “The device of claim 1, wherein the device further comprises a digital processor adapted to control the operation of the at least one first LPHDR execution unit.”	213
H. Claim 26: “The device of claim 1, wherein the device is part of a mobile device.”	216
I. Claim 28: “The device of claim 1, wherein the at least one first LPHDR execution unit represents numbers using a floating point representation.”	216
VII. CLAIMS 1-2, 21-24, 26, 28, AND 32-33 WOULD HAVE BEEN OBVIOUS OVER DOCKSER AND TONG	216
A. Tong (Ex. 1008).....	216
B. Claims 1-2, 21-24, 26, and 28	224
C. Claim 32: “The device of claim 1, wherein the device is adapted to perform nearest neighbor search.”	228
D. Claim 33	230
VIII. CLAIMS 1-26, 28, 36-61, AND 63 WOULD HAVE BEEN OBVIOUS OVER DOCKSER AND MACMILLAN	236
A. MacMillan (Ex. 1009)	236
B. Dockser/MacMillan Combination	240
C. Claims 1 and 28	245
D. Claims 3, 7, and 9	247
E. Claims 5, 8, and 10	248
F. Claims 2, 4, and 6	255
G. Claims 11-17.....	258
H. Claim 18: “The device of claim 8, wherein the dynamic range of the possible valid inputs to the first operation is at least as wide as from 1/1,000,000 through 1,000,000.”	263
I. Claim 19: “The device of claim 1, wherein the at least one first LPHDR execution unit comprises a plurality of locally connected LPHDR execution units.”	263

J.	Claim 20: “The device of claim 1, wherein the device has a SIMD architecture.”	266
K.	Claim 21: “The device of claim 1, wherein the device includes memory locally accessible to the at least one first LPHDR execution unit.”	267
L.	Claims 22-23.....	268
M.	Claim 24: “The device of claim 1, wherein the device further comprises a digital processor adapted to control the operation of the at least one first LPHDR execution unit.”	270
N.	Claim 25	273
O.	Claim 26: “The device of claim 1, wherein the device is part of a mobile device.”	274
P.	Claims 36-61 and 63.....	274
IX.	CLAIMS 1-26, 28, 32-61, 63, AND 67-70 WOULD HAVE BEEN OBVIOUS OVER DOCKSER, TONG, AND MACMILLAN	275
A.	Claims 1-26, 28, 32, 36-61, 63, and 67	275
B.	Claims 33-35 and 68-70	278
C.	Alternative Interpretation of Claims 5-6, 8, 10-18, and 35-70.....	279
X.	APPENDIX I – TECHNICAL DETAILS OF RELATIVE ERROR ANALYSES.....	286
A.	Dockser’s Multiplier Relative Error Is Independent of Exponent and Sign	286
B.	Software Demonstration of Dockser’s Register Bit-Dropping	290
C.	Algebraic Analysis of Dockser’s Register Bit-Dropping.....	297
D.	Software Demonstration of Dockser’s Logic Bit-Dropping	303
E.	Adjustment to Account for Overflow/Underflow	316

DETAILED TABLE OF CONTENTS

I.	PERSONAL AND PROFESSIONAL BACKGROUND	1
II.	MATERIALS REVIEWED AND CONSIDERED	3
III.	MY UNDERSTANDING OF PATENT LAW	5
	A. Anticipation	7
	B. Obviousness.....	7
IV.	THE '273 PATENT	10
	A. Overview	10
	B. Claims	20
	C. Person of Ordinary Skill in the Art	21
	D. Prosecution History	24
V.	BATES-2010 RENDERS OBVIOUS CLAIMS 1-70.....	24
	A. Bates-2010 Is Prior Art Because the Challenged Claims Are Not Entitled to the '201 Application's Priority Date	24
	B. <u>Written Description</u> : The '201 Application Does Not Demonstrate Possession of the Full Scope of Any of the Challenged Claims	25
	1. The Challenged Claims Broadly Cover a Genus of "Execution Units" Defined by Its Functional Performance.....	25
	a. The Claims Recite Any "Execution Unit" Adapted to Execute Any "Operation"	25
	b. The Claims Specify Functional Performance Characteristics of the "Execution Unit" Genus Rather Than Structural Features Common to Members of That Genus.....	35
	2. The '201 Application's Disclosure Does Not Demonstrate Dr. Bates Possessed the Full Scope of Any Challenged Claim	41
	a. The '201 Application's Description of a Conventional Digital Silicon Transistor-Based Implementation	44
	b. The Digital Silicon Transistor-Based Implementation Is Not Representative of the Vast Genus of Execution Units Claimed	53

(1) The Digital Silicon Transistor-Based Implementation Is Not Representative of Analog Execution Units Implemented in Silicon.....	53
(2) The Digital Silicon Transistor-Based Implementation Is Not Representative of Execution Units Implemented Using Unpredictable and Nascent Technologies.....	62
(3) Conclusion: The Digital Silicon Transistor-Based Implementation Is Not Representative of the Claimed Genus	67
c. The '201 Application Does Not Disclose Structural Features Common to the Claimed Genus	68
C. <u>Enablement</u> : The '201 Application Does Not Enable the Full Scope of Any Challenged Claim.....	69
1. Breadth of the Claims and Nature of the Invention	69
2. State of the Art, Level of POSA's Skill, Level of Unpredictability in the Art, and Quantity of Experimentation Needed to Practice the Alleged Invention Based on the Disclosure	71
3. Amount of Direction Provided, Including Working Examples, in the Specification.....	73
D. Bates-2010 Renders Obvious Claims 1-70	74
1. Bates-2010's Status As Prior Art	74
2. An Obvious Implementation of Bates-2010's Silicon Transistor-Based Logarithmic Execution Unit	75
3. Bates-2010 Renders Obvious Claim 1 of the '273 patent.....	79
a. [1A1] A device comprising: at least one first low precision high dynamic range (LPHDR) execution unit	79
b. [1A2] adapted to execute a first operation on a first input signal representing a first numerical value to produce a first output signal representing a second numerical value,.....	82
c. [1B1] wherein the dynamic range of the possible valid inputs to the first operation is at least as wide as from 1/65,000 through 65,000	86

d. [1B2] for at least $X=5\%$ of the possible valid inputs to the first operation, the statistical mean, over repeated execution of the first operation on each specific input from the at least $X\%$ of the possible valid inputs to the first operation, of the numerical values represented by the first output signal of the LPHDR unit executing the first operation on that input differs by at least $Y=0.05\%$ from the result of an exact mathematical calculation of the first operation on the numerical values of that same input; and	87
4. Claims 3, 5, 7-8, 9-10	91
5. Claims 2, 4, 6	94
6. Claims 11-17	96
7. Claim 18. The device of claim 8, wherein the dynamic range of the possible valid inputs to the first operation is at least as wide as from 1/1,000,000 through 1,000,000	97
8. Claim 19. The device of claim 1, wherein the at least one first LPHDR execution unit comprises a plurality of locally connected LPHDR execution units	97
9. Claim 20. The device of claim 1, wherein the device has a SIMD architecture	98
10. Claim 21. The device of claim 1, wherein the device includes memory locally accessible to the at least one first LPHDR execution unit	98
11. Claims 22-23	99
12. Claim 24. The device of claim 1, wherein the device further comprises a digital processor adapted to control the operation of the at least one first LPHDR execution unit.....	100
13. Claim 25	103
14. Claim 26. The device of claim 1, wherein the device is part of a mobile device.	104
15. Claim 27. The device of claim 1, wherein the at least one first LPHDR execution unit represents numbers using a logarithmic representation	104

16. Claim 28. The device of claim 1, wherein the at least one first LPHDR execution unit represents numbers using a floating point representation	105
17. Claim 29. The device of claim 1: wherein the device further comprises input means for receiving data representing an input image; and wherein the input image includes the first input signal.....	105
18. Claim 30. The device of claim 29, wherein the device is part of a mobile device.....	109
19. Claim 31. The device of claim 29, wherein the device is adapted to deblur the input image	110
20. Claim 32. The device of claim 1, wherein the device is adapted to perform nearest neighbor search	110
21. Claims 36-67	111
22. Claims 33 and 68.....	116
23. Claims 34-35, 69-70	120
VI. CLAIMS 1-2, 21-24, 26, AND 28 WOULD HAVE BEEN OBVIOUS OVER DOCKSER.....	122
A. Dockser (Ex. 1007).....	123
B. Claim 1	131
1. [1A1] A device comprising: at least one first low precision high dynamic range (LPHDR) execution unit.....	131
2. [1A2] Adapted to execute a first operation on a first input signal representing a first numerical value to produce a first output signal representing a second numerical value.....	145
3. [1B1] wherein the dynamic range of the possible valid inputs to the first operation is at least as wide as from 1/65,000 through 65,000	160
4. [1B2] for at least X=5% of the possible valid inputs to the first operation, the statistical mean, over repeated execution of the first operation on each specific input from the at least X% of the possible valid inputs to the first operation, of the numerical values represented by the first output signal of the LPHDR unit executing the first operation on that input differs by at least	

Y=0.05% from the result of an exact mathematical calculation of the first operation on the numerical values of that same input;....	163
a. The “Statistical Mean” Limitation	163
b. The “Exact Mathematical Calculation” Limitation	167
c. The Relative Error Amount (Y) for a Fraction of the Possible Valid Inputs (X) Limitation.....	168
(1) Dockser’s Register Bit-Dropping Meets Limitation [1B2]	183
(i) Software Demonstration That Dockser’s Register Bit-Dropping Meets Limitation [1B2]	194
(ii) Pencil-and-Paper Algebraic Demonstration That Dockser’s Register Bit-Dropping Meets Limitation [1B2].....	195
(2) Dockser’s Multiplier Logic Bit-Dropping Meets Limitation [1B2]	196
(3) Performing Register and Multiplier Logic Bit- Dropping Together Also Meets [1B2].....	206
C. Claim 2: “The device of claim 1, wherein the at least one first LPHDR execution unit comprises at least part of an FPGA”	209
D. Claim 21: “The device of claim 1, wherein the device includes memory locally accessible to the at least one first LPHDR execution unit.”	209
E. Claim 22: “The device of claim 1, wherein the device is implemented on a silicon chip”	212
F. Claim 23: “The device of claim 1, wherein the device is implemented on a silicon chip using digital technology.”	213
G. Claim 24: “The device of claim 1, wherein the device further comprises a digital processor adapted to control the operation of the at least one first LPHDR execution unit.”	213
H. Claim 26: “The device of claim 1, wherein the device is part of a mobile device.”	216
I. Claim 28: “The device of claim 1, wherein the at least one first LPHDR execution unit represents numbers using a floating point representation.”	216

VII. CLAIMS 1-2, 21-24, 26, 28, AND 32-33 WOULD HAVE BEEN OBVIOUS OVER DOCKSER AND TONG216

 A. Tong (Ex. 1008).....216

 B. Claims 1-2, 21-24, 26, and 28224

 C. Claim 32: “The device of claim 1, wherein the device is adapted to perform nearest neighbor search.”228

 D. Claim 33230

 1. [33A1] “A device comprising a computer processor and a computer-readable memory storing computer program instructions, wherein the computer program instructions are executable by the processor to emulate a second device comprising: a plurality of components comprising: at least one first low precision high-dynamic range (LPHDR) execution unit”231

 2. Limitations [33A2]-[33B2]235

VIII. CLAIMS 1-26, 28, 36-61, AND 63 WOULD HAVE BEEN OBVIOUS OVER DOCKSER AND MACMILLAN236

 A. MacMillan (Ex. 1009)236

 B. Dockser/MacMillan Combination240

 C. Claims 1 and 28245

 D. Claims 3, 7, and 9247

 E. Claims 5, 8, and 10248

 F. Claims 2, 4, and 6255

 G. Claims 11-17.....258

 H. Claim 18: “The device of claim 8, wherein the dynamic range of the possible valid inputs to the first operation is at least as wide as from 1/1,000,000 through 1,000,000.”263

 I. Claim 19: “The device of claim 1, wherein the at least one first LPHDR execution unit comprises a plurality of locally connected LPHDR execution units.”263

 J. Claim 20: “The device of claim 1, wherein the device has a SIMD architecture.”266

K.	Claim 21: “The device of claim 1, wherein the device includes memory locally accessible to the at least one first LPHDR execution unit.”	267
L.	Claims 22-23.....	268
M.	Claim 24: “The device of claim 1, wherein the device further comprises a digital processor adapted to control the operation of the at least one first LPHDR execution unit.”	270
N.	Claim 25	273
O.	Claim 26: “The device of claim 1, wherein the device is part of a mobile device.”	274
P.	Claims 36-61 and 63.....	274
IX.	CLAIMS 1-26, 28, 32-61, 63, AND 67-70 WOULD HAVE BEEN OBVIOUS OVER DOCKSER, TONG, AND MACMILLAN	275
A.	Claims 1-26, 28, 32, 36-61, 63, and 67	275
B.	Claims 33-35 and 68-70	278
C.	Alternative Interpretation of Claims 5-6, 8, 10-18, and 35-70.....	279
X.	APPENDIX I – TECHNICAL DETAILS OF RELATIVE ERROR ANALYSES.....	286
A.	Dockser’s Multiplier Relative Error Is Independent of Exponent and Sign	286
B.	Software Demonstration of Dockser’s Register Bit-Dropping	290
C.	Algebraic Analysis of Dockser’s Register Bit-Dropping.....	297
D.	Software Demonstration of Dockser’s Logic Bit-Dropping	303
E.	Adjustment to Account for Overflow/Underflow	316

I, Richard Goodin, declare:

1. I have been retained by Wolf, Greenfield & Sacks, P.C., counsel for Petitioner Google LLC, to assess claims 1-70 (the “challenged claims”) of U.S. Patent No. 8,407,273 (“the ’273 patent”). I am being compensated for my time at my standard rate of \$500.00 per hour, plus actual expenses. My compensation is not dependent in any way upon the outcome of the *inter partes* review of the ’273 patent.

I. PERSONAL AND PROFESSIONAL BACKGROUND

2. I have over 40 years of experience in computing, including computer architecture. I graduated from the University of Delaware in 1978 with a bachelor’s degree in electrical engineering with a minor in mechanical engineering. Over the next 20 years, I designed and implemented computer graphics software and hardware for companies including Apple and Sun Microsystems. Since 1990, I have worked as a consultant designing, implementing, and testing graphics hardware for various companies and for the United States military.

3. In the course of my education and work experience, I have become very familiar with the design of microprocessors, including the design of arithmetic execution units for microprocessors. The ability to efficiently and rapidly execute large numbers of arithmetic operations is extremely important to the field of computer graphics. My experience in improving graphics performance

entails an understanding of the ways in which computers represent and perform calculations on numbers, and of the tradeoffs that come with different numerical representations and calculation techniques. It also entails an understanding of how to design and program hardware that can perform calculations in parallel. I have used my knowledge in both of these areas throughout my career. For example, at Raydiant, Inc. I was the chief hardware architect for an advanced graphics accelerator, and at Sun Microsystems I programmed many system graphics components such as the windowing system and graphics APIs for a massively parallel graphics processing hardware architecture called Jet.

4. I am a Senior Member of the Institute of Electrical and Electronics Engineers (“IEEE”) and a Senior Member of the Association for Computing Machinery (“ACM”). The IEEE is a professional society for electrical and electronics engineering whose core purpose is to foster technological innovation and excellence for the benefit of humanity. For admission to the grade of Senior Member, a candidate shall be an engineer in IEEE-designated fields for a total of 10 years and have demonstrated five years of significant performance. The ACM is the world’s largest educational and scientific computing society. Senior Member is an earned membership grade awarded to approximately 25% of members who have demonstrated performance that sets them apart from their

peers. I am also a licensed Professional Engineer in the state of North Carolina, Registration Number 036347.

5. My curriculum vitae is provided as Exhibit 1004.

II. MATERIALS REVIEWED AND CONSIDERED

6. My findings, as explained below, are based on my years of education, experience, and background in the field of computing, as well as my investigation and study of relevant materials for this declaration. When developing the opinions set forth in this declaration, I assumed the perspective of a person having ordinary skill in the art, as set forth in Section IV.C below. In forming my opinions, I have studied and considered the materials identified in the list below.

Exhibit	Description
1001	U.S. Patent No. 8,407,273
1002	Prosecution History of U.S. Patent No. 8,407,273
1005	U.S. Patent Appl. 12/816,201 (“201 Application”)
1006	U.S. Patent Appl. Publ. No. 2010/0325186 A1 (“Bates-2010”)
1007	U.S. Patent App. Publ. No. 2007/0203967 (“Dockser”)
1008	Tong et. al, <i>Reducing Power by Optimizing the Necessary Precision/Range of Floating-Point Arithmetic</i> , IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 8, No. 3, June 2000 (“Tong”)
1009	U.S. Patent No. 5,689,677 (“MacMillan”)
1010	U.S. Patent Appl. Publ. No. 2007/0266071 (“Dockser-Lall”)
1011	U.S. Patent No. 6,065,209 (“Weiss”)
1012	Gaffar et. al, <i>Unifying Bit-width Optimization for Fixed-Point and Floating-Point Designs</i> , 12 th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, April 20-23, 2004 (“Gaffar”)
1013	European Patent Appl. Publ. No. 0 632 369 A1 (“Hekstra”)
1014	U.S. Patent No. 5,375,084 (“Begun”)

1015	U.S. Patent No. 4,933,895 (“Grinberg”)
1016	U.S. Patent No. 5,442,577 (“Cohen”)
1017	U.S. Patent Appl. Publ. No. 2003/0028759 (“Prabhu”)
1018	U.S. Patent No. 5,790,834 (“Dreyer”)
1019	U.S. Patent Appl. Publ. No. 2009/0066164 (“Flynn”)
1020	U.S. Patent No. 5,666,071 (“Hawkins”)
1021	A Matter of Size: Triennial Review of the National Nanotechnology Initiative (National Academies Press 2006), pages 15-44, 99-109
1023	U.S. Patent No. 6,311,282 (“Nelson”)
1024	U.S. Patent No. 4,583,222 (“Fossum”)
1029	U.S. Patent Appl. Publ. No. 2007/0033572 (“Donovan”)
1030	U.S. Patent No. 5,623,616 (“Vitale”)
1031	David A. Patterson and John L. Hennessy, <i>Computer Organization and Design</i> (Morgan Kaufmann 3 rd ed. Revised 2007) (“Patterson”), pages 189-217
1042	Prosecution History of U.S. Patent No. 8,150,902, issued from U.S. Patent Appl. No. 12/816,201
1047	Thomas Way et. al, <i>Compiling Mechanical Nanocomputer Components</i> , Global Journal of Computer Science and Technology, Vol. 10, Issue 2 (Ver. 1.0), April 2010, pp. 36-42 (“Way”)
1048	David Nield, <i>In a Huge Milestone, Engineers Build a Working Computer Chip out of Carbon Nanotubes</i> , Sciencealert.com, Dec. 7, 2019 (accessed Sep. 9, 2020) (“Nield”)
1049	Leah Cannon, <i>What Can DNA-Based Computers Do?</i> , MIT Technology Review, Feb. 4, 2015 (accessed Sep. 8, 2020) (“Cannon”)
1050	Katherine Bourzac, <i>The First Carbon Nanotube Computer</i> , MIT Technology Review, Sep. 25, 2013 (accessed Sep. 8, 2020) (“Bourzac”)
1051	Joe Touch et. al., <i>Optical Computing</i> , Nanophotonics 2017 6(3): 503-505 (“Touch”)
1052	Robert Allen, Ed., <i>The Penguin Complete English Dictionary</i> , (Penguin 2006) (“Penguin”), page 1411, definition of “supercomputer”
1053	<i>A Dictionary of Computing</i> (Oxford 6 th ed. 2008) (“Oxford”), page 500, definition of “supercomputer”
1054	U.S. Patent Appl. Pub. No. 2006/0270110 (“Steffen”)
1055	U.S. Patent Appl. Pub. No. 2009/0188705 (“Kacker”)
1056	U.S. Patent Appl. Pub. No. 2007/0050566 (“Lang”)
1057	U.S. Patent No. 6,622,135 (“Tremiolles”)
1058	U.S. Patent Appl. Pub. No. 2005/0235070 (“Young”)
1059	U.S. Patent No. 7,301,436 (“Hopper”)

1060	U.S. Patent Appl. Pub. No. 2003/0204760 (“Youngs”)
1061	U.S. Patent No. 10,416,961 (“’961 Patent”)
1062	U.S. Patent No. 9,218,156 (“’156 Patent”)

III. MY UNDERSTANDING OF PATENT LAW

7. In developing my opinions, I discussed various relevant legal principles with Petitioner’s attorneys. Though I do not purport to have prior knowledge of such principles, I understood them when they were explained to me and have relied upon such legal principles, as explained to me, in the course of forming the opinions set forth in this declaration. My understanding in this respect is as follows:

8. I understand that “*inter partes* review” (IPR) is a proceeding before the United States Patent & Trademark Office for evaluating the patentability of an issued patent’s claims based on prior-art patents and printed publications.

9. I understand that, in this proceeding, Petitioner has the burden of proving that the challenged claims of the ’273 patent are unpatentable by a preponderance of the evidence. I understand that “preponderance of the evidence” means that a fact or conclusion is more likely true than not true.

10. I understand that, in IPR proceedings, claim terms in a patent are given their ordinary and customary meaning as understood by a person of ordinary skill in the art (“POSA”) in the context of the entire patent and the prosecution history pertaining to the patent. If the specification provides a special definition

for a claim term that differs from the meaning the term would otherwise possess, the specification's special definition controls. If a claim element is expressed as a "means" for performing a specified function, I understand that it covers the corresponding structure described in the specification and equivalents of the described structure. I have applied these standards in preparing the opinions in this declaration.

11. I understand that determining whether a particular patent or printed publication constitutes prior art to a challenged patent claim can require determining the effective filing date (also known as the priority date) to which the challenged claim is entitled. I understand that for a patent claim to be entitled to the benefit of the filing date of an earlier application to which the patent claims priority, the earlier application must have described the claimed invention in sufficient detail to convey with reasonable clarity to the POSA that the inventor had possession of the claimed invention as of the earlier application's filing date. I understand that a disclosure that merely renders the claimed invention obvious is not sufficient written description for the claim to be entitled to the benefit of the filing date of the application containing that disclosure.

12. I understand that for an invention claimed in a patent to be patentable, it must be, among other things, new (novel—*i.e.*, not anticipated) and not obvious

from the prior art. My understanding of these two legal standards is set forth below.

A. Anticipation

13. I understand that, for a patent claim to be “anticipated” by the prior art (and therefore not novel), each and every limitation of the claim must be found, expressly or inherently, in a single prior-art reference. I understand that a claim limitation is disclosed for the purpose of anticipation if a POSA would have understood the reference to disclose the limitation based on inferences that a POSA would reasonably be expected to draw from the express teachings in the reference when read in light of the POSA’s knowledge and experience.

14. I understand that a claim limitation is inherent in a prior art reference if that limitation is necessarily present when practicing the teachings of the reference, regardless of whether a person of ordinary skill recognized the presence of that limitation in the prior art.

B. Obviousness

15. I understand that a patent claim may be unpatentable if it would have been obvious in view of a single prior-art reference or a combination of prior-art references.

16. I understand that a patent claim is obvious if the differences between the subject matter of the claim and the prior art are such that the subject matter as a

whole would have been obvious to a person of ordinary skill in the relevant field at the time the invention was made. Specifically, I understand that the obviousness question involves a consideration of:

- the scope and content of the prior art;
- the differences between the prior art and the claims at issue;
- the knowledge of a person of ordinary skill in the pertinent art; and
- if present, objective factors indicative of non-obviousness, sometimes referred to as “secondary considerations.” As of the time of this declaration, I am unaware of the Patent Owner having asserted any such secondary considerations with respect to the ’273 patent.

17. I understand that in order for a claimed invention to be considered obvious, a POSA must have had a reason for combining teachings from multiple prior-art references (or for altering a single prior-art reference, in the case of obviousness in view of a single reference) in the fashion proposed.

18. I further understand that in determining whether a prior-art reference would have been combined with other prior art or with other information within the knowledge of a POSA, the following are examples of approaches and rationales that may be considered:

- combining prior-art elements according to known methods to yield predictable results;
- simple substitution of one known element for another to obtain predictable results;
- use of a known technique to improve similar devices in the same way;
- applying a known technique to a known device ready for improvement to yield predictable results;
- applying a technique or approach that would have been “obvious to try,” *i.e.*, choosing from a finite number of identified, predictable solutions, with a reasonable expectation of success.
- known work in one field of endeavor may prompt variations of it for use in either the same field or a different one based on design incentives or other market forces if the variations would have been predictable to one of ordinary skill in the art;
- some teaching, suggestion, or motivation in the prior art that would have led one of ordinary skill to modify the prior-art reference or to combine prior-art reference teachings to arrive at the claimed invention. I understand that this teaching, suggestion or

motivation may come from a prior-art reference or from the knowledge or common sense of one of ordinary skill in the art.

19. I understand that for a single reference or a combination of references to render the claimed invention obvious, a POSA must have been able to arrive at the claimed invention by altering or combining the applied references.

IV. THE '273 PATENT

A. Overview

20. The '273 patent claims a device comprising at least one “low precision high-dynamic range (LPHDR) execution unit.” '273 patent, claim 1. The claimed “execution unit” is adapted to execute an operation (*e.g.*, an arithmetic operation like multiplication) on an input signal representing a first numerical value to produce an output signal representing a second numerical value. *See, e.g.*, '273 patent, 29:65-30:3 (claim 1: “A device comprising: at least one first low precision high dynamic range (LPHDR) execution unit adapted to execute a first operation on a first input signal representing a first numerical value to produce a first output signal representing a second numerical value”). The “execution unit” is characterized as “LPHDR” because it performs a “low precision” (“LP”) operation on a “high dynamic range” (“HDR”) of values.

21. Precision relates to accuracy and can be impacted by the ability to represent numbers that differ from each other by small amounts—*e.g.*, fractional

values. For example, in a decimal (base-10) representation, the number 2.13 (represented using two fractional digits) is more precise than if rounded to 2.1 (using just one fractional digit), which is more precise than if rounded to 2 (using no fractional digits).

22. In a binary “fixed-point” representation, any fixed number of bits can be chosen to represent a number’s fractional component. For example, a binary fixed-point representation that allocates three bits to representing a fractional component can represent numbers to the nearest $1/8$, which is more precise than a representation that allocates only two bits to the fractional component and can represent numbers to the nearest $1/4$, which is more precise than a representation that allocates no bits to the fractional component and can only represent integers.

23. A “low precision arithmetic” operation, as the ’273 patent says, is an operation that is less precise than an exact calculation, such that “each operation might introduce” some “error” in its “results.” ’273 patent, 4:9-12 (describing prior art processors that “provide low precision arithmetic, in which each operation might introduce perhaps an error of a few percentage points in its results”). For example, a low-precision operation may produce a result of 2.1 when the exact operation would produce a result of 2.13.

24. “Dynamic range” refers to the range of numerical values that a signal can represent or a unit can operate on or produce. *See* ’273 patent, 2:35-39 (“In

some embodiments, the processing elements have ‘high dynamic range’ in the sense that they are capable of operating on inputs and/or producing outputs spanning a range at least as large as from one millionth to one million.”). For example, a signal that can represent any numerical value between 1 and 100 has a higher dynamic range than one that can only represent numerical values between 1 and 10.

25. In a binary (base-2) representation of integers, the more bits used to represent a number, the wider the possible dynamic range—*e.g.*, two bits can represent the numbers 0-3, three bits can represent the numbers 0-7, etc. Other binary representations have been developed that can have a wider dynamic range for the same number of bits. One such well-known representation is floating-point representation.

26. For example, in describing a prior art “16 bit floating point representation,” the ’273 patent quotes a 2008 Wikipedia article that explains:

This format is used in several computer graphics environments including OpenEXR, OpenGL, and D3DX. ***The advantage over 8-bit or 16-bit binary integers is that the increased dynamic range*** allows for more detail to be preserved in highlights and shadows. The advantage over 32-bit single precision binary formats is that it requires half the storage and bandwidth.

'273 patent, 5:23-30.¹ Similarly, Tong (Ex. 1008) explains that a 32-bit floating-point number in the “IEEE single-precision” format has a much wider dynamic range than a 32-bit integer:

For instance, an IEEE single-precision number and an integer on a 32-bit machine both require 32 bits of storage for their representation.

However, the dynamic range provided by the FP representation ($1.99999988 \times 2^{-126}$ to $1.99999988 \times 2^{127}$) is substantially wider than that provided by the integer representation ($1-2^{31}$).

Tong, 274.

27. It was well known before 2009 that for a numerical representation using a fixed number of bits, a tradeoff is necessary between dynamic range and precision. For example, a ten-bit binary fixed-point representation that allocates all ten bits to the non-fractional part, and thus represents values between zero and 1,023 but only to the nearest integer, has higher dynamic range but lower precision than a representation that allocates all ten bits to representing the fractional part and thus only represents values between zero and one but is accurate to the nearest 1/1024.

28. Another example relates to the well-known floating-point representations, which can represent a high dynamic range of numerical values using a smaller number of digits than a fixed-point representation—*e.g.*, by

¹ All emphases in this declaration are added unless otherwise indicated.

representing a large number as a smaller number (called the “mantissa”) scaled by an order of magnitude. A simple example of a floating-point representation of a decimal number using base ten would represent the number 3,046,000 as 3.046×10^6 .

29. In a typical binary (base-2) floating-point representation, one bit is allocated to indicate the number’s sign (positive or negative), some bits are allocated to specify the “exponent” (the number’s order of magnitude in base two), and other bits are allocated to specify the mantissa. *See, e.g.,* Dockser (Ex. 1007), [0001] (“To find the value of a floating-point number, the mantissa is multiplied by ***a base (commonly 2 in computers)*** raised to the power of the exponent.”), [0002] (“the ANSI/IEEE-754 standard (commonly followed by modern computers) specifies a 32-bit single format having a 1-bit sign, an 8-bit exponent, and a 23-bit mantissa.”); Tong (Ex. 1008), 274 (“There are two different IEEE standards for FP computation. IEEE 754 is a binary standard that requires the implied radix (base) to be two. IEEE 854 allows either radix 2 or radix 10 representation. By far, IEEE 754 is more popular and most desktop microprocessors support the IEEE 754 standard.”).

30. It was known that the number of exponent bits in a floating-point representation impacts dynamic range, that the number of mantissa (fraction) bits impacts precision, and that there was a tradeoff:

The designer of a floating-point representation must find a compromise between the size of the [mantissa] fraction and the size of the exponent because a fixed word size means you must take a bit from one to add a bit to the other. This trade-off is between precision and range: increasing the size of the [mantissa] fraction enhances the precision of the fraction, while increasing the size of the exponent increases the range of numbers that can be represented.

Patterson (Ex. 1031), 191.

31. It was known in mathematics and in computing that LPHDR arithmetic had benefits. For example, Tong (Ex. 1008) noted that “wide dynamic range” was recognized as “a desirable feature,” and explained that “[i]t has long been known that many... applications can get by with less precision” over that range. Tong, 273. Tong explains that “Floating-point (FP) hardware provides a wide dynamic range of representable real numbers,” and notes that “FP hardware’s performance, simplified programming model, and adaptability over a wide dynamic range makes it a desirable feature.” Tong, 273. Tong states that its experimental results show that “programs such as speech recognition and image processing use significantly less power with our reduced bitwidth FP representation than with an IEEE-standard FP representation.” Tong, 273. Tong also notes that “[i]t has long been known that many such signal processing applications can get by with less precision/range than full FP,” *i.e.*, than with the an “IEEE-standard FP representation.” Tong, 273. Tong’s own experiments

demonstrated that “the FP format provides essential dynamic range... but the fine precision of the 23-bit mantissa is not essential.” Tong, 279.

32. Tong showed experimentally that certain applications could function properly with low-precision HDR arithmetic (“the fine precision of the 23-bit mantissa is not essential”), and that lowering precision saved power by “reduc[ing] waste [from] unnecessary bits.” Tong, 273, 277-279. Tong’s authors “examine how software can employ the *minimal* number of mantissa and exponent bits in FP hardware to *reduce* power consumption, yet *maintain* a program’s overall accuracy.” Tong, 273 (emphasis in original). Tong’s “fundamental principle is to reduce waste—in this case, unnecessary bits in the FP representation and computation.” Tong, 273.

33. Tong demonstrated that “reduction in the mantissa bitwidth is the most effective means of reducing power in FP [floating point] datapath elements,” such as multipliers. Tong, 277. Tong describes an experiment “[t]o determine the impact of different mantissa and exponent bitwidths” by “emulat[ing] in software different bitwidth FP [floating point] units.” Tong, 278.

34. The authors used these emulated FP units on “a set of five signal processing applications” and ran these programs “across a range of mantissa bitwidths.” Tong, 278. This experiment showed that “[n]one of the workloads display a noticeable degradation in accuracy when the mantissa bitwidth is reduced

from 23 to 11 bits,” and that for two of the applications, “the accuracy does not change significantly with as few as 5 mantissa bits.” Tong, 278. As Tong explains,

The reason behind this result is that many programs dealing with human interfaces process sensory data with intrinsically low resolutions. Raw input data with 4-10 bits of precision is rather common in these applications. While intermediate results often require more dynamic range than is available with small bitwidth fixed-point computation, the programs do not require vastly more precision. This is different from scientific programs such as large-scale computational fluid dynamics or electrical circuit simulation, which not only require a huge amount of precision and dynamic range but also delicate rounding modes to preserve the accuracy of the results. *What is quite clear from these experiments is that the FP format provides essential dynamic range* (we can reduce, but not reduce dramatically, the number of exponent bits) *but the fine precision of the 23-bit mantissa is not essential* (half as many bits often suffice)

Tong, 278-279.

35. Similarly, Dockser (Ex. 1007), published in 2007, disclosed a low-precision HDR execution unit that saved power by reducing the precision at which it performed floating-point operations to whatever precision was needed for a particular application. See Dockser, [0003]-[0007], which I discuss in Section

VI.A below. The challenged claims of the '273 patent encompass this prior-art concept, as I explain in Sections VI-IX below.

36. Claim 1 of the '273 patent recites that the claimed “execution unit” operates on a dynamic range of possible valid inputs that is “at least as wide as from 1/65,000 through 65,000.” '273 patent, 30:4-6. Claim 1 also recites that, for some minimum claimed percentage of the possible inputs (*i.e.*, at least 5%), the outputs the unit generates differ on average by a claimed minimum amount (*i.e.*, at least 0.05%) from the result of an “exact mathematical calculation” of that operation on “the numerical values” of those “same input[s].” '273 patent, 30:6-16 (“for at least $X=5\%$ of the possible valid inputs to the first operation, the statistical mean ... of the numerical values represented by the first output signal of the LPHDR unit executing the first operation on that input differs by at least $Y=0.05\%$ from the result of an exact mathematical calculation of the first operation on the numerical values of that same input”).

37. The challenged claims thus cover a genus of “execution units” defined by a functional performance characteristic—executing an operation with a specified degree of imprecision on a high dynamic range of inputs. The specification makes clear that the claims cover any implementation of an “execution unit” that performs any operation using any technology. As I discuss in more detail in Section V.B.2.b below, the specification includes as an aspirational

wish-list of technologies that its inventor speculated “may” someday “enable other sorts of traditional digital and analog computing processors or other devices” beyond traditional silicon-implemented processors, and alleges that these technologies could be used in some undescribed way to implement a high dynamic range execution unit that meets the claimed imprecision functional performance characteristics. *See, e.g.*, ’273 patent, 26:17-31 (“Although certain embodiments of the present invention are described herein as being implemented using silicon chip fabrication technology, this is merely an example and does not constitute a limitation of the present invention. Alternatively, for example, embodiments of *the present invention may be implemented* using technologies that *may enable other sorts* of traditional digital and analog computing processors or other devices. Examples of such technologies include various nanomechanical and nanoelectronic technologies, chemistry based technologies such as for DNA computing, nanowire and nanotube based technologies, optical technologies, mechanical technologies, biological technologies, and other technologies whether based on transistors or not that are *capable of implementing LPHDR architectures* of the kinds disclosed herein.”).

B. Claims

38. I have been asked to provide my opinions regarding claims 1-70 of the '273 patent. Independent claim 1 is reproduced below, annotated to label claim elements.

[1A1] A device comprising: at least one first low precision high dynamic range (LPHDR) execution unit

[1A2] adapted to execute a first operation on a first input signal representing a first numerical value to produce a first output signal representing a second numerical value,

[1B1] wherein the dynamic range of the possible valid inputs to the first operation is at least as wide as from $1/65,000$ through $65,000$ and

[1B2] for at least $X=5\%$ of the possible valid inputs to the first operation, the statistical mean, over repeated execution of the first operation on each specific input from the at least $X\%$ of the possible valid inputs to the first operation, of the numerical values represented by the first output signal of the LPHDR unit executing the first operation on that input differs by at least $Y=0.05\%$ from the result of an exact mathematical calculation of the first operation on the numerical values of that same input

Limitations [1A1]-[1A2] recite a device comprising an LPHDR execution unit adapted to execute an operation. [1B1] recites a claimed dynamic range of the inputs, (" $1/65,000$ through $65,000$ "), and [1B2] recites a claimed minimum

imprecision of the operation in terms of relative error (Y) (“Y=0.05%”) produced for a percentage (X) (“X=5%”) of inputs.

C. Person of Ordinary Skill in the Art

39. I have been informed and understand that for purposes of assessing whether prior-art references disclose every element of a patent claim (thus “anticipating” the claim) and/or would have rendered the claim obvious, the patent and the prior-art references must be assessed from the perspective of a person having ordinary skill in the art (“POSA”) to which the patent is related, based on the understanding of that person at the time of the patent claim’s priority date. I have been informed and understand that a POSA is presumed to be aware of all pertinent prior art and the conventional wisdom in the art, and is a person of ordinary creativity. I have applied this standard throughout my declaration.

40. The ’273 patent involves technology in the field of computing. In relation to Sections VI-IX below (Dockser-based grounds), I have been asked to provide my opinions as to the state of the art in this field by 2009. I use this timeframe because the face of the ’273 patent indicates an earliest claimed priority date of the June 19, 2009 filing date of a provisional application. ’273 patent (Ex. 1001), (60). Whenever I offer an opinion in relation to the Dockser-based grounds (Sections VI-IX) below about the knowledge of a POSA, the manner in which a POSA would have understood the claims of the ’273 patent or its description, the

manner in which a POSA would have understood the prior art, or what a POSA would have been led to do based on the prior art, I am referencing the 2009 timeframe, even if I do not say so specifically in each case.

41. I understand that the Patent Owner may attempt to prove that the alleged invention recited in the challenged claims was conceived at some time prior to the earliest claimed priority date on the face of the patent (June 19, 2009). At the time of this declaration, I am unaware of the Patent Owner having alleged any earlier conception date or produced any evidence to establish any earlier conception date.

42. In relation to Section V (Bates-2010-based ground) below, I have been asked to provide my opinions as to the state of the art by 2010, because the '273 patent claims priority to a non-provisional patent application No. 12/816,201 ("the '201 Application") filed on June 15, 2010. '273 patent, (63). Whenever I offer an opinion in relation to the Bates-2010-based ground (Section V) below about the knowledge of a POSA, the manner in which a POSA would have understood the claims of the '273 patent or its description, the manner in which a POSA would have understood the description of the '201 Application and of Bates-2010, the manner in which a POSA would have understood the prior art, or what a POSA would have been led to do based on the prior art (including Bates-2010), I am referencing the 2010 timeframe, even if I do not say so specifically in

each case. Demonstrating that the '273 patent's claims would have been obvious in 2010 demonstrates that they remained obvious any time after 2010, because the POSA's level of skill only increases over time.

43. In my opinion, a POSA in 2009 or 2010 would have had at least a bachelor's degree in Electrical Engineering, Computer Engineering, Applied Mathematics, or the equivalent, and at least two years of academic or industry experience in computer architecture. More education could substitute for experience, and vice versa. This person would have been capable of understanding and applying the teachings of the prior-art references discussed in this declaration, and the teachings of the '273 patent to the extent of the written description and enablement it provides.

44. By 2009, I held a Bachelor's degree in Electrical Engineering, and I had 30 years of industry experience in computer architecture, including the design of microprocessors and parallel processing systems. Therefore, I was a person of more than ordinary skill in the art during the relevant time period. However, I worked with many people who fit the characteristics of the POSA, and I am familiar with their level of skill. When developing the opinions set forth in this declaration, I assumed the perspective of a person having ordinary skill in the art, as set forth above.

45. If it is determined that the '273 patent is only entitled to its 2012 filing date ('273 patent, (22)), a POSA in 2012 would have had the same or greater level of skill as in 2009 or 2010 because the POSA's level of skill only increases over time, so the challenged claims would have been obvious for the same reasons I discuss in this Declaration.

D. Prosecution History

46. I have reviewed the prosecution history of the '273 patent (Ex. 1002). The examiner allowed the claims of the '273 patent without substantively discussing any prior art. The examiner simply said "the prior art does not teach or suggest at least" the "wherein" clause recited in the independent claims. '273 FH (Ex. 1002), pages 164-167.

47. The '273 patent claims priority to the '201 Application. I have reviewed the file history of the '201 Application. In that case, the examiner also allowed the claims without substantively discussing any prior art. *See* Ex. 1042 (application 12/816,201), pages 164-166.

V. BATES-2010 RENDERS OBVIOUS CLAIMS 1-70

A. Bates-2010 Is Prior Art Because the Challenged Claims Are Not Entitled to the '201 Application's Priority Date

48. According to the face of the document, U.S. Patent Application Publication No. 2010/0325186 (Ex. 1005) is the published version of the '201 Application, published on December 23, 2010. Based on my review, the

disclosures in the '273 patent (Ex. 1001), the '201 Application as originally filed (Ex. 1005), and the published version of the '201 Application (Ex. 1006) are substantively identical other than their claims. I refer to the published version of the '201 Application (U.S. Patent Application Publication No. 2010/0325186, Ex. 1006) as “Bates-2010.”

B. Written Description: The '201 Application Does Not Demonstrate Possession of the Full Scope of Any of the Challenged Claims

1. The Challenged Claims Broadly Cover a Genus of “Execution Units” Defined by Its Functional Performance

a. The Claims Recite Any “Execution Unit” Adapted to Execute Any “Operation”

49. Each challenged claim in the '273 patent recites an “execution unit adapted to execute a first operation on a first input signal representing a first numerical value to produce a first output signal representing a second numerical value.” *See* '273 patent, 29:66-30:3 (independent claim 1), 31:37-41 (independent claim 33), 31:63-67 (independent claim 36), 34:5-9 (independent claim 68).

50. The '273 patent does not define the claim term “execution unit” or tie it to any particular structure. In the entire specification of the patent, the term “execution unit” appears only twice. First, the specification describes how “execution units” are “drive[n]” in an example of prior art “array processors,” but it does not describe an execution unit’s structure in that example. *See* '273 patent, 3:53-56 (“Array processors gain speed by using silicon efficiently—using just one

instruction fetch/decode unit to drive many small simple execution units in parallel.”). Later, when describing its own embodiments, the patent states: “references herein to ‘processing elements’ within embodiments of the present invention should be understood more generally as any kind of execution unit, whether for performing LPHDR operations or otherwise.” ’273 patent, 8:7-11.

51. Rather than defining any particular structure, the claims characterize the recited “execution unit” by what it does (execute an operation on a high-dynamic-range input) and how accurately it does it (with low precision).

52. The ’273 patent’s specification says an “execution unit” can be implemented using digital circuits that operate on digital representations (using binary 0s and 1s) of numerical values (*e.g.*, 11:53-56), analog circuits operating on analog representations of numerical values (*e.g.*, 14:16-26), or combinations of the two (24:47-57). At 11:53-56, the patent states: “One *digital embodiment* of the LPHDR arithmetic unit 408 *operates on digital (binary) representations of numbers*. In one digital embodiment these numbers are represented by their logarithms.” The “LPHDR arithmetic unit” referred to in that passage is part of a “processing element” that the patent says is an example of an “execution unit.” *See* ’273 patent, 8:7-11 (“references herein to ‘processing elements’ within embodiments of the present invention should be understood more generally as *any*

kind of execution unit, whether for performing LPHDR operations or otherwise”). At 14:16-26, the patent states:

Some *embodiments of the present* invention may include *analog* representations and processing methods. Such embodiments may, for example, *represent LPHDR values as charges, currents, voltages, frequencies, pulse widths, pulse densities, various forms of spikes, or in other forms not characteristic of traditional digital implementations*. There are many such representations discussed in the literature, along with mechanisms for processing values so represented. Such methods, often called Analog methods, can be used to perform LPHDR arithmetic in the broad range of architectures we have discussed, of which SIMD is one example.

’273 patent, 14:16-26. Finally, the patent says that “embodiments of the present invention may represent values in any variety of ways, such as by using digital or analog representations,” (24:47-49), and that “LPHDR arithmetic circuits may be implemented in any of a variety of ways, such as by using various digital methods (which may be parallel or serial, pipelined or not) or analog methods or combinations thereof” (24:54-57).

53. The ’273 patent’s specification also describes using numerous types of digital and analog representations of numbers, including but not limited to (i.e., “such as”), “fixed point, logarithmic, or floating point” digital representations (24:49-50) and analog representations taking the form of “charges, currents,

voltages, frequencies, pulse widths, pulse densities, various forms of spikes, or in *other forms* not characteristic of traditional digital implementations” (14:16-26; *see also* 24:50-53. These representations “may be used individually or in combination.” ’273 patent, 24:52-54.

54. The specification also says that while certain embodiments may be implemented using transistors and “silicon chip fabrication technology,” “this is merely an example and does not constitute a limitation of the present invention.” ’273 patent, 26:17-20. Instead, the patent states that “the present invention *may* be implemented using technologies that *may enable* other sorts of traditional digital and analog computing processors” such as “various nanomechanical and nanoelectronic technologies,” or “chemistry based technologies such as for [sic] DNA computing” or “nanowire and nanotube based technologies, ... and *other [unstated] technologies whether based on transistors or not* that are capable of implementing LPHDR architectures.” ’273 patent, 26:17-31. Independent claim 6 of related U.S. Patent No. 10,416,961 (“the ’961 patent”) recites a “wherein” clause making clear that the generic “execution unit” in the independent claims of both the ’961 patent and the ’273 patent encompasses these technologies. ’961 patent (Ex. 1061), 31:5-13; *see also* Ex. 1062 (related U.S. Patent No. 9,218,156, “the ’156 patent”), 30:44-53, 31:59-32:4.

55. Along similar lines, the “operation” the execution unit performs includes not only relatively simple arithmetic operations (’273 patent, 11:44-48 (“addition, multiplication, subtraction, and division”)), but also more complex operations such as “trigonometric functions” (*id.*, 27:33-39) and even “non-linear operations such as exponentiation” (*id.*, 1:65).

56. Additionally, the claims expressly cover *non-deterministic* implementations, *i.e.*, execution units that produce different results for different executions of the same operation on the same input. All of the challenged independent claims recite “repeated execution” of the operation on “that same input” and taking a “statistical mean” (which a POSA would have understood is an average) of the output numerical values. *See* claims 1, 33, 36, and 68. As I explain in paragraphs 57-60 below, a POSA would have understood that the claims expressly cover non-deterministic embodiments via the recited “statistical mean” provisions.

57. The specification says some embodiments (*e.g.*, analog embodiments) are non-deterministic. In discussing the prior art, the patent describes prior art “[a]rray processors” that “use analog representations of numbers and analog circuits to perform computations,” and states that the “**SCAMP**” computer “is such a machine.” ’273 patent, 4:7-9. The patent notes that “[t]hese machines...introduce noise into their computations, so the computations are not

repeatable.” ’273 patent, 4:9-13. Later, when describing its embodiments, the patent states that “[s]ome embodiments of the present invention may include analog representations and processing methods.” ’273 patent, 14:16-17. The patent states that “[s]uch methods, often called Analog methods, can be used to perform LPHDR arithmetic in the broad range of architectures we have discussed, of which SIMD is one example,” and then states that “[a]n example of an *analog SIMD* architecture is the *SCAMP* design (and related designs) of *Dudek*,” where “values have low dynamic range” and are “accurate roughly to within 1%.” ’273 patent, 14:23-30. The patent later purportedly describes “how to build an analog SIMD machine that performs LPHDR arithmetic, and is thus an embodiment of the present invention,” ’273 patent, 14:50-52, and states that in this purported machine, “[t]he analog value may be accurate to about 1%, following the approach of *Dudek*,” ’273 patent, 14:57-61. Thus, a POSA would have understood that the patent’s “analog” embodiment is based on the SCAMP design by Dudek, and hence that the analog embodiment is non-deterministic because its “computations are not repeatable.” ’273 patent, 4:13.

58. The specification even alleges that some *digital* embodiments “may not yield deterministic... results.” ’273 patent, 26:42-46 (“For certain embodiments of the present invention, even if implemented using only digital

techniques, the arithmetic operations may not yield deterministic, repeatable, or the most accurate possible results within the chosen low precision representation.”).

59. It was well-known, however, that a digital computing circuit is typically deterministic, meaning that if it repeats the same operation on the same input, it produces the same output—*e.g.*, in a typical digital computing circuit, adding the bit sequence 011 to the bit sequence 010 always produces the output sequence 101. For evidence corroborating that this was background knowledge for a POSA, *see, e.g.*, Weiss (Ex. 1011), 1:40-42 (“Moreover, conventional digital circuits are traditionally deterministic. Thus, the output of a conventional digital circuit is typically predictable.”). The ’273 patent explains that its claimed LPHDR execution unit covers not only such “repeatable” deterministic embodiments (’273 patent, 17:10-14, 26:42-46), but also analog embodiments that are non-deterministic because they “introduce noise into their computations, so the computations are not repeatable.” ’273 patent, 4:7-13; *see also* my discussion in paragraph 57 above (citing ’273 patent, 4:7-13, 14:16-61). *See also* ’273 patent, 17:10-14.

60. As an example of non-deterministic arithmetic, if an analog circuit represents the number 1 as a voltage somewhere between 0.99 and 1.01 (’273 patent, 14:13-30), then the output when adding 1+1 may be 1.98 the first time, 2.01 the second time, etc. A POSA would have understood that for such non-

deterministic embodiments, the limitation in the '273 patent's independent claims of the "statistical mean, over repeated execution of the first operation on each specific input..., of the numerical values represented by the first output signal of the LPHDR unit executing the first operation on that input" is met by averaging (taking the statistical mean of) the different outputs produced by the same operation on the same input. For instance, the mean of 1.98 and 2.01 is 1.995 in the above example.

61. Based on all of the different possibilities for implementation technologies, number representations, and operations that the '273 patent's specification discusses (see my discussion in paragraphs 52-60 above), a POSA would have understood that the claimed "execution unit" in the '273 patent ostensibly covers a large number of types of potential implementations. According to the patent, each "execution unit" implementation must involve at least one selection from each of the at least three classes of variables I discuss in paragraphs 62-64 below.

62. The first variable is the **implementing technology**, *i.e.*, the technology used to construct a physical execution unit. The '273 patent says that each "execution unit" is implemented either using one of nine technologies that the patent identifies—"silicon," "nanomechanical," "nanoelectrical," "DNA," "nanowire," "nanotube," "optical," "mechanical," "biological"—or using some

“*other*” technologies that the patent says could be used, but doesn’t say what they are. *See* ’273 patent, 26:17-31 (“Although certain embodiments of the present invention are described herein as being implemented using silicon chip fabrication technology, this is merely an example ... embodiments of the present invention may be implemented using technologies that may enable other sorts of traditional digital and analog computing processors or other devices. Examples of such technologies include various nanomechanical and nanoelectronic technologies, chemistry based technologies such as for DNA computing, nanowire and nanotube based technologies, optical technologies, mechanical technologies, biological technologies, and other technologies whether based on transistors or not that are capable of implementing LPHDR architectures of the kinds disclosed herein.”).

63. The second variable is the **numerical representation**. The ’273 patent says that each execution unit is adapted to execute either (i) fixed point, logarithmic, or floating-point digital representations of numbers, (ii) analog representations of numbers such as “charges,” “voltages,” “various forms of spikes,” or “other forms,” or (iii) a combination of digital and analog representations. *See* ’273 patent, 11:53-56 (“One digital embodiment of the LPHDR arithmetic unit 408 operates on *digital (binary)* representations of numbers. In one digital embodiment these numbers are represented by their *logarithms*.”), 14:16-26 (“Some embodiments of the present invention may include

analog representations and processing methods. Such embodiments may, for example, represent LPHDR values as *charges, currents, voltages, frequencies, pulse widths, pulse densities, various forms of spikes*, or in other forms not characteristic of traditional digital implementations.”), 24:47-57 (“...embodiments of the present invention may represent values in any of a variety of ways, such as by using *digital or analog representations, such as fixed point, logarithmic, or floating point representations*, voltages, currents, charges, pulse width, pulse density, frequency, probability, spikes, timing, or combinations thereof. These underlying representations may be used individually or in combination to represent the LPHDR values. LPHDR arithmetic circuits may be implemented in any of a variety of ways, such as by using various *digital methods* (which may be parallel or serial, pipelined or not) *or analog methods or combinations thereof*.”).

64. The third variable is the **operation**. The '273 patent says each execution unit is adapted to execute at least one operation, which could be any operation ranging from addition to multiplication to a trigonometric operation or exponentiation, etc. *See, e.g.*, '273 patent, 1:63-66 (mentioning “interesting non-linear operations, such as exponentiation” that could be performed after solving the alleged deficiencies in the prior art), 11:44-48 (“Other collections of LPHDR operations may be used to approximate LPHDR arithmetic operations, such as addition, multiplication, subtraction, and division, using techniques that are well-

known to those having ordinary skill in the art.”), 27:31-40 (“...consider an embodiment in which LPHDR arithmetic elements can perform one or more operations (perhaps including, for example, trigonometric functions) ... and for each specific set of input values the LPHDR elements each produce one or more output values (for example, simultaneously computing both sin and cos of an input)...”).

65. Because the '273 patent does not restrict how the three variables I discussed above can be combined, there are an immense number of ways to combine them, and thus an immense number of different types of execution units that the claims purport to cover. For example, according to the specification, the claims would purportedly cover DNA-implemented and nanomechanical-implemented execution units operating on analog representations of numbers to perform trigonometric operations; silicon-implemented execution units operating on analog representations (e.g., pulse densities) to perform multiplication; and execution units implemented using some “*other*” unspecified technology operating on analog representations using some “*other*” unspecified physical characteristic to perform a non-linear operation.

b. The Claims Specify Functional Performance Characteristics of the “Execution Unit” Genus Rather

Than Structural Features Common to Members of That Genus

66. As I discussed above (Section IV.A), the '273 patent's claimed "execution unit" is limited only by functional performance characteristics regarding high dynamic range and low precision. The independent claims' requirement that the "execution unit" perform an operation on a **high dynamic range** of numbers (*i.e.*, a range "at least as wide as from 1/65,000 through 65,000") was not novel; the '273 patent concedes that execution units that performed precise operations on high-dynamic-range numbers were known. For example, the patent states that prior art processors "deliver great precision, performing exact arithmetic with integers typically 32 or 64 bits long and performing rather accurate and widely standardized arithmetic with 32 and 64 bit floating point numbers." '273 patent, 3:15-18. The patent later refers to the "large dynamic range of typical 32 or 64 bit floating point representations." '273 patent, 4:13-17. Thus, a POSA would have understood that 32 and 64 bit floating point numbers were examples of high (*i.e.* "large") dynamic range numbers on which "rather accurate" arithmetic was performed. *See also* Tong (Ex. 1008), 274 (explaining that the well-known "IEEE single-precision" 32-bit format has a "dynamic range" of " $1.99999988 \times 2^{-126}$ to $1.99999988 \times 2^{127}$ ").

67. The independent claims' requirement that the "execution unit" perform its operation with **low precision** is recited in terms of the percentage of

inputs (“at least $X=5\%$ ”) for which the output of the operation differs, on average (the claimed “statistical mean”), by a specific percentage (“at least $Y=0.05\%$ ”) from an exact mathematical calculation of the operation, as can be seen from the highlighted limitations below.

Claim 1
[1A1] A device: comprising at least one first low precision high dynamic range (LPHDR) execution unit
[1A2] adapted to execute a first operation on a first input signal representing a first numerical value to produce a first output signal representing a second numerical value,
[1B1] wherein the dynamic range of the possible valid inputs to the first operation is at least as wide as from $1/65,000$ through $65,000$ and
[1B2] for at least $X=5\%$ of the possible valid inputs to the first operation, the statistical mean, over repeated execution of the first operation on each specific input from the at least $X\%$ of the possible valid inputs to the first operation, of the numerical values represented by the first output signal of the LPHDR unit executing the first operation on that input differs by at least $Y=0.05\%$ from the result of an exact mathematical calculation of the first operation on the numerical values of that same input.

Claim 33
[33A1] A device comprising a computer processor and a computer-readable memory storing computer program instructions, wherein the computer program instructions are executable by the processor to emulate a second device comprising: at least one first low precision high-dynamic range (LPHDR) execution unit
[33A2] adapted to execute a first operation on a first input signal representing a first numerical value to produce a first output signal representing a second numerical value;
[33B1] wherein the dynamic range of the possible valid inputs to the first operation is at least as wide as from $1/65,000$ through $65,000$ and

[33B2] for at least $X=5\%$ of the possible valid inputs to the first operation, the statistical mean, over repeated execution of the first operation on each specific input from the at least $X\%$ of the possible valid inputs to the first operation, of the numerical values represented by the first output signal of the LPHDR unit executing the first operation on that input differs by at least $Y=0.05\%$ from the result of an exact mathematical calculation of the first operation on the numerical values of that same input.

Claim 36

[36A1] A device: comprising at least one first low precision high-dynamic range (LPHDR) execution unit

[36A2] adapted to execute a first operation on a first input signal representing a first numerical value to produce a first output signal representing a second numerical value,

[36B1] wherein the dynamic range of the possible valid inputs to the first operation is at least as wide as from $1/65,000$ through $65,000$ and

[36B2] for at least $X=5\%$ of the possible valid inputs to the first operation, the statistical mean, over repeated execution of the first operation on each specific input from the at least $X\%$ of the possible valid inputs to the first operation, of the numerical values represented by the first output signal of the LPHDR unit executing the first operation on that input differs by at least $Y=0.05\%$ from the result of an exact mathematical calculation of the first operation on the numerical values of that same input;

[36C] wherein the number of LPHDR execution units in the device exceeds the non-negative integer number of execution units in the device adapted to execute at least the operation of multiplication on floating point numbers that are at least 32 bits wide.

Claim 68

[68A1] A device comprising a computer processor and a computer-readable memory storing computer program instructions, wherein the computer program instructions are executable by the processor to emulate a second device comprising: at least one first low precision high-dynamic range (LPHDR) execution unit

[68A2] adapted to execute a first operation on a first input signal representing a first numerical value to produce a first output signal representing a second numerical value,

[68B1] wherein the dynamic range of the possible valid inputs to the first operation is at least as wide as from $1/65,000$ through $65,000$ and

[68B2] for at least 5% of the possible valid inputs to the first operation, the statistical mean, over repeated execution of the first operation on each specific input from the at least 5% of the possible valid inputs to the first operation, of the numerical values represented by the first output signal of the LPHDR unit executing the first operation on that input differs by at least 0.05% from the result of an exact mathematical calculation of the first operation on the numerical values of that same input;

[68C] wherein the number of LPHDR execution units in the second device exceeds the non-negative integer number of execution units in the second device adapted to execute at least the operation of multiplication on floating point numbers that are at least 32 bits wide.

68. It can be seen above that the “at least” X and Y percentages characterizing the execution unit’s imprecision in the independent claims have no upper bound. For example, even an “execution unit” that executes an operation on high-dynamic-range inputs with 100% of its calculations differing from the exact mathematical calculation by 500% or more is within the scope of the claims. The claimed X and Y only recite minimum values required to be met or exceeded in order to meet the claims.

69. Like the claimed high dynamic range, the claimed low precision was also not novel. The ’273 patent acknowledges that low-precision high-dynamic-range execution units were known, but alleges they were considered “not useful.” ’273 patent, 6:57-7:11 (“First, it is commonly believed by those having ordinary skill in the art, that LPHDR computation, and in particular massive amounts of LPHDR computation, whether performed in a massively parallel way or not, is not

practical as a substrate for moderately general computing. Second, it is commonly believed by those having ordinary skill in the art that massive amounts of even high precision computation on a single chip or in a single machine, as is enabled by a compact arithmetic processing unit, is not useful without a corresponding increase in bandwidth between processing elements within the machine and into and out of the machine because computing is wire limited and arithmetic can be considered to be available at no cost. Despite these views—that massive amounts of arithmetic on a chip or in a massively parallel machine are not useful, and that massive amounts of LPHDR arithmetic are even worse—embodiments of the present invention disclosed herein demonstrate that massively parallel LPHDR designs are in fact useful and provide significant practical benefits in at least several significant applications”).

70. The '273 patent's claimed “execution unit” is thus characterized as a “*low precision* high dynamic range (LPHDR) execution unit” (29:66-67 (claim 1)) because for at least a certain percentage of calculations the output is at least a certain percent different from the “exact mathematical” answer. The challenged claims cover any “execution unit adapted to execute” any high-dynamic-range operation with the recited minimum degree of imprecision.

2. The '201 Application's Disclosure Does Not Demonstrate Dr. Bates Possessed the Full Scope of Any Challenged Claim

71. The '201 Application says that both the “degree of precision” and the “frequency” with which an execution unit “may yield only approximation”—both determining whether an execution unit satisfies the recited “X” and “Y” functional performance characteristics in the '273 patent’s claims—“vary from implementation to implementation.” '201 Application (Ex. 1005), [0142] (“The degree of precision of a ‘low precision, high dynamic range’ arithmetic element may vary from implementation to implementation. ... [A] LPHDR arithmetic element may produce results which are sometimes (or all of the time) no closer than 1%, or 2%, or 5%, or 10%, or 20% to the correct result.”); *id.*, [0144] (“The frequency with which LPHDR arithmetic elements may yield only approximations to correct results may vary from implementation to implementation. ... [C]onsider further a fraction F of the valid inputs and a relative error amount E by which the result calculated by an LPHDR element may differ from the mathematically correct result. ... In certain embodiments ... F is 1% In several other example embodiments, F is not 1 % but instead is one of 2%, or 5%, or 10%, or 20%, or 50%.”). A POSA would have understood that the “LPHDR arithmetic element” referred to in these passages is an LPHDR execution unit, because it *executes* operations, *i.e.* mathematical operations. *See* '201 Application, [0144] (“consider

an embodiment in which LPHDR arithmetic elements can perform one or more operations (perhaps including, for example, trigonometric functions))), [0042] (“references herein to ‘processing elements’ within embodiments of the present invention should be understood more generally as *any kind of execution unit, whether for performing LPHDR operations* or otherwise”).

72. The ’201 Application is thus candid that the claimed “execution unit” can be any device with any structure that implements the abstract idea of executing operations with low precision. However, the ’201 Application does not explain how to achieve the claimed degrees of imprecision or frequencies, nor does it describe a particular structure that achieves those degrees of imprecision or frequencies. See my discussion in Sections V.B.2.a-V.B.2.c below. In my opinion, therefore, a POSA would not have understood the ’201 Application as demonstrating the inventor’s possession of an execution unit meeting the claimed requirements, because the ’201 Application fails to tie the claimed imprecision level (the ’273 patent’s claims’ recited “X” and “Y” percentages) to any specific implementation described in the application. Additionally, even if the ’201 Application had demonstrated possession of a species of the claimed execution unit implemented using conventional digital circuitry, in my opinion a POSA would not have understood the ’201 Application as demonstrating the inventor’s full possession of the full scope of the claimed subject matter, because the application

does not demonstrate possession of any of the other types of execution units covered by the '273 patent's claims. For example, the '201 Application does not provide any design details for how to implement the types of execution units mentioned in paragraph [0139] (e.g., a "nanomechanical" implementation, a "DNA" implementation, or an "other [unspecified] technologies" implementation), let alone one that is capable of executing *any* mathematical operation within the claimed functional performance characteristics ("X" and "Y").

73. The '201 Application does little more than describe the known abstract idea that performing imprecise computer operations was beneficial in certain applications. The '201 Application alleges that "today's CPU chips make inefficient use of their transistors" because "conventional CPUs typically are designed to provide [the] precision" provided by 32 or 64 bit floating point numbers, "using on the order of a million transistors to implement the arithmetic operations require "on the order of a million transistors to implement ... arithmetic operations." '201 Application, [0020]. The Application then claims that "[t]here are many economically important applications ... which are not especially sensitive to precision and that would greatly benefit ... from the ability to draw upon a far greater fraction of the computing power inherent in those million transistors," but that "[c]urrent architectures for general purpose computing fail to deliver this power." '201 Application, [0021].

74. But the idea that imprecise operations were beneficial in some applications was well-known years before the '201 Application was filed. For example, Tong (Ex. 1008), published in 2000, noted that although “wide dynamic range” was recognized as “a desirable feature,” “[i]t has long been known that many... applications can get by with less precision.” Tong, 273. *See also* my further discussion of Tong’s teachings in Section VII below. The '201 Application ([0144]) identifies numerous specific possible values for a minimum amount of error (Y) and a minimum percentage of valid inputs (X) resulting in that amount of error, but I find (and in my opinion a POSA would have found) nothing in the '201 Application establishing that any specific one(s) of the listed possible combinations of X and Y values are any more critical than any of the other possible combinations.

a. The '201 Application’s Description of a Conventional Digital Silicon Transistor-Based Implementation

75. As I explain below, the LPHDR “execution unit” implementation that the '201 Application describes with the most specificity is one using silicon transistor-based circuits that operate on digital representations of numbers. *See* Bates-2010, [0035], [0061]-[0062]. That is unsurprising as this was conventional technology that had already been used to implement an LPHDR execution unit. *See* my discussion of Dockser and Tong in Sections VI-IX below.

76. The '201 Application mentions an execution unit using a floating-point digital representation of numbers having a 10-bit mantissa, 6-bit exponent and 1-bit sign, and characterizes that floating-point execution unit as “represent[ing] values from one millionth up to one million with a precision of about 0.1%.” '201 Application, [0035] (“One variety of LPHDR arithmetic represents values from one millionth up to one million with a precision of about 0.1%. If these values were represented and manipulated using the methods of floating point arithmetic, they would have binary mantissas of no more than 10 bits plus a sign bit and binary exponents of at least 5 bits plus a sign bit.”).

77. Other than that reference in paragraph [0035], a floating-point execution unit is not described anywhere in the specification. The '201 Application does not explain what it means by “precision of about 0.1%” and does not tie it to the '273 patent’s claimed error (Y) percentage for a particular minimum percentage (X) of possible valid inputs. Even if the “precision of about 0.1%” ([0035]) were considered to describe the claimed error amount (Y) characteristic (which is not clear), the '201 Application says nothing about the percentage of valid inputs that would have this amount of error in the floating-point execution unit, so it does not describe a floating-point execution unit where $X \geq 5\%$ as claimed. The '201 Application provides no explanation of how to

implement a floating-point execution unit to ensure that it has the claimed functional performance characteristics (X and Y).

78. As I discuss below, the '201 Application describes an execution unit that operates on logarithmic numeric representations in more detail. *See* '201 Application, [0035], [0061]-[0074], FIGS. 4-6. After mentioning float-point operations in passing, the '201 Application states that “[o]ne example of an alternative embodiment is to use a logarithmic representation of the values.” '201 Application, [0035]. The discussed logarithmic representation “is called a Logarithmic Number System (LNS),” which the Application admits was “well-understood by those having ordinary skill in the art.” '201 Application, [0061].

79. The '201 Application focuses on this “logarithmic” embodiment, devoting fourteen paragraphs to discussing it. The '201 Application states that it uses LNS in “[o]ne digital embodiment of the LPHDR arithmetic unit 408;” in fact, the LNS embodiment is the **only** digital embodiment of that unit that the '201 Application describes. '201 Application, [0061]. The '201 Application explains that in an LNS, “[f]or numbers to have representation errors of at most, say, 1% (one percent), the fractional part of this logarithm should be represented with enough precision that the least possible change in the fraction corresponds to about a 1% change in the value” of a number. '201 Application, [0062]. The '201 Application states that “in this example embodiment”—*i.e.*, the “digital

embodiment of the LPHDR arithmetic unit 408” ([0061])—“the fraction part of the representation has 6 bits.” ’201 Application, [0062]. The ’201 Application also states that using 6 bits for the fractional part “means that numbers may be represented in the present embodiment with a multiplicative error of approximately 1%,” but does not explain what it means by “multiplicative error,” or how the number of fraction bits produces this amount of “multiplicative error.” ’201 Application, [0062].

80. In addition, “[i]n the present embodiment,” *i.e.*, the LNS embodiment, “the integer part of the logarithm representation is 5 bits long,” which the ’201 Application explains is enough to “represent numbers whose absolute value is from, say, one billionth to one billion.” ’201 Application, [0063].

81. The number format in the LNS embodiment includes three more bits in addition to the 5 integer bits and 6 fraction bits. One of these is a “sign bit in the exponent” (’201 Application, [0063]), which “permit[s] the logarithm to be negative” (’201 Application, [0066]); a POSA would have understood from basic mathematics that negative logarithms are needed to represent numbers with a magnitude less than 1. The representation also includes a bit representing the “sign of the value” of the overall number, and a “Not a Number (NaN)” bit that is “set if some problem has arisen in computing the value.” ’201 Application, [0065]-[0066]. Figure 5 shows the “word format 500 for these numbers”—*i.e.* numbers

represented using the logarithmic representation—“in the present embodiment.”

'201 Application, [0066].

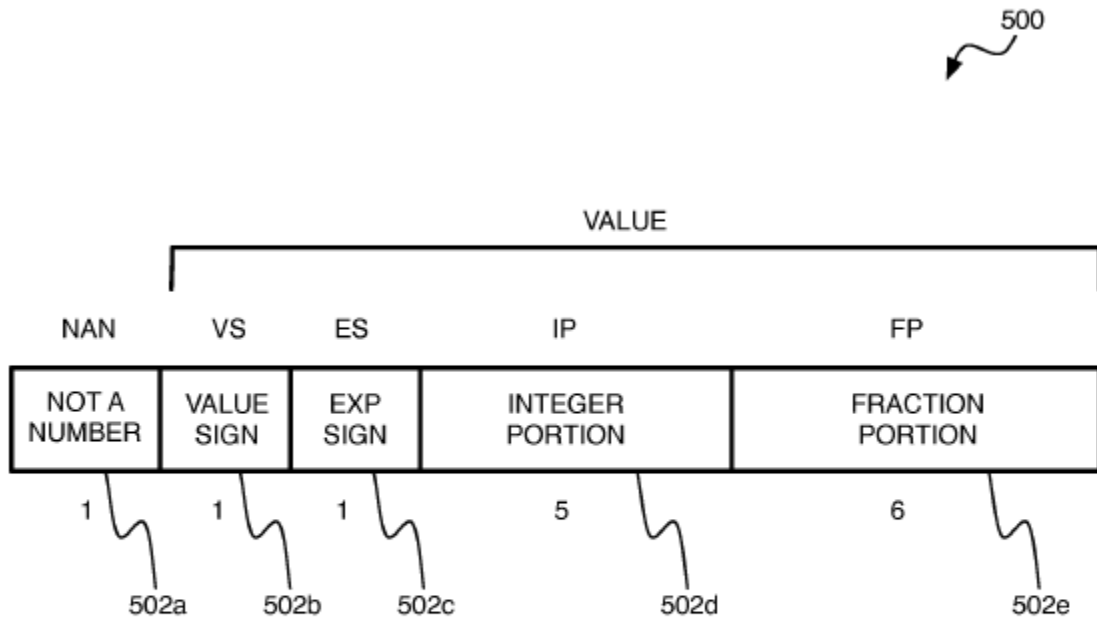


FIG. 5

82. Figure 6 of the '201 Application “shows an example digital implementation of the LPHDR arithmetic unit 408 for the representation illustrated in FIG. 5”—*i.e.*, a digital implementation of the LNS embodiment. '201 Application, [0067]. The '201 Application describes this unit, and also discusses how it performs certain calculations, in paragraphs [0067]-[0074].

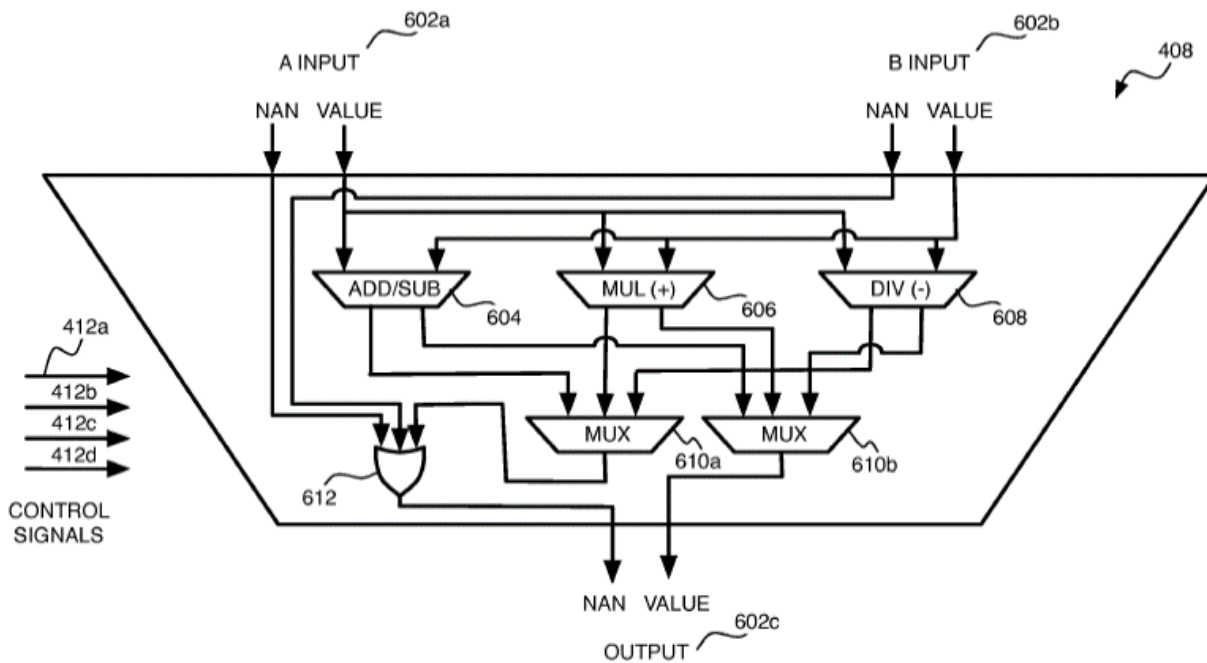


FIG. 6

83. Although the '201 Application describes this implementation as having “a multiplicative error of approximately 1%” ([0062]), it does not explain what this means and does not tie it to the specific error (Y) and frequency (X) imprecision characteristics claimed. Even if this “multiplicative error of approximately 1%” were somehow considered to support the claimed minimum error of “at least $Y=0.05\%$,” the '201 Application says nothing about the percentage of valid inputs that would have this amount of error and thus does not describe a logarithmic execution unit meeting “at least $X=5\%$ ” as claimed.

84. The '201 Application also describes a software “model” that simulates the logarithmic embodiment using “repeatable” “arithmetic” that “produces errors

of up to approximately 1-2% in each operation.” ’201 Application, [0085] (“For the goal of showing usefulness, we choose a very general embodiment of an LPHDR machine. Our model of the machine is that it provides at least the following capabilities: (1) is massively parallel, (2) provides LPHDR arithmetic possibly with noise ... This model includes, among others, implementations that are digital or analog or mixed, have zero or more noise ...”), [0086] (“Applications that meet both requirements running in this model will function well on many kinds of LPHDR machines, and thus those machines are a broadly useful invention.”), [0087] (“Applications are tested using two embodiments for the machine’s arithmetic. ...”), [0088] (“A second embodiment uses logarithmic arithmetic with a value representation as shown in FIG. 5. The arithmetic is repeatable, that is, not noisy, but because of the short fraction size it produces errors of up to approximately 1-2% in each operation. In the following discussion, this embodiment is denoted ‘lns’”).

85. In describing this software simulation, once again, the ’201 Application says nothing about what percentage of valid inputs would produce the minimum relative error required by each of the ’273 patent’s claims. The ’201 Application reports a “mean score error” ([0104]-[0105], [0107]) from a number of test runs comparing the results of executing certain high-level software programs using this model against the results of performing those same software programs

using “high precision” arithmetic, but as I explain further below, a POSA would have understood that those results show the error in the high-level program’s results, not the amount of error in an individual *operation* as the challenged claims require. ’201 Application, [0090]-[0110].

86. The ’201 Application describes a known mathematical problem known as the “nearest neighbor problem (‘NN’)” and purports to “show that approximate nearest neighbor is computable using embodiments of the present invention.” ’201 Application, [0090], [0094]. The ’201 Application describes an “algorithm which may be performed by machines implemented according to embodiments of the present invention” that takes as input “a set of Examples and a Test vector” and “seeks to find the nearest (or almost nearest) Example to the Test.” ’201 Application, [0095]; *see also* algorithm and software description at [0096]-[0104]. Each “test run” “generated one hundred Test vectors,” and “[f]or each Test, the nearest neighbor was *computed both according to the enhanced algorithm* above and *according to the standard nearest neighbor method* using high precision floating point arithmetic.” ’201 Application, [0104]. Although paragraph [0104] refers to the “fp+noise” model of an analog embodiment (which I discuss in Section V.B.2.b(1) below), the ’201 Application states that “[a] similar computation was then performed using ‘lns’ arithmetic” ([0107]); thus, a POSA would have understood that the discussion of the test in paragraph [0104] also

applies to the simulation of the logarithmic embodiment, except in terms of which type of LPHDR arithmetic was purportedly modeled. The '201 Application reports results for the nearest neighbor test using the logarithmic embodiment, including “mean score error,” in paragraph [0107]. The '201 Application explains that the “mean score error” for the “nearest neighbor” program refers to the error in a “score” produced by the program. '201 Application, [0105] (“The ‘mean score error’ values are considered below in the discussion of weighted scores.”), [0116] (“The C code described above computes weighted scores along with nearest neighbors. ... the code performs a number of runs, each producing many Examples and Tests, and compares results of traditional floating point computations with results calculated using fp+noise and lns arithmetic.”).

87. A POSA would have understood that producing the above “score” in the nearest neighbor program can involve *millions* of individual operations, and a POSA would have understood that the amount of error in any individual operation cannot be discerned from the average error in the final score produced. '201 Application, [0101]-[0110]. The '201 Application explains that “computing an overall weighted score involves summing the individual weighted scores associated with each Example,” and that “[s]ince each run was processing 1,000,000 Examples, this means that the sums were over one million small positive values.” '201 Application, [0118]. Thus, a POSA would have understood that the

“mean score error” that the ’201 Application reports does *not* refer to error for individual arithmetic operations, or even individual weighted scores, but rather is the mean error for the *sum* of “over one million small positive values” that were themselves computed by another series of arithmetic operations. Nowhere in the discussion of its test results (*see* ’201 Application, [0101]-[0117]) does the ’201 Application report error percentages for individual arithmetic operations, or explain how one can derive those percentages from the mean score errors reported.

88. For the above reasons I have explained in this section, in my opinion a POSA would have understood that the ’201 Application does not describe any execution unit—even one implemented with conventional technology (*i.e.*, silicon transistor-based circuits) and operating on digital representations of numbers—as meeting any challenged claim of the ’273 patent.

b. The Digital Silicon Transistor-Based Implementation Is Not Representative of the Vast Genus of Execution Units Claimed

(1) The Digital Silicon Transistor-Based Implementation Is Not Representative of Analog Execution Units Implemented in Silicon

89. The ’273 patent’s claims encompass silicon-implemented execution units operating on analog representations of numbers such as “charges, currents, voltages, frequencies, pulse widths, pulse densities, various forms of spikes, or in other forms not characteristic of traditional digital implementations,” ’273 patent,

14:16-26, as I discussed in Section V.B.1.a above. Analog implementations rely on “the natural analog physics of transistors or other physical devices instead of using only the digital subset of the device’s behavior.” ’201 Application, [0036]. Analog implementations are more unpredictable than digital implementations and can be non-deterministic (i.e., an analog execution unit may produce different results at different times for the same input value). *See, e.g.*, ’201 Application, [0026] (prior art “SCAMP” processors “introduce noise into their computations, so the computations are not repeatable”), [0141] (“For certain embodiments of the present invention, *even if implemented using only* digital techniques”—implying that this would also be true of analog techniques—“the arithmetic operations may not yield deterministic, repeatable, or the most accurate possible results within the chosen low precision representation.”).

90. Analog computing using silicon circuits was not a nascent technology in 2010. As the ’201 Application mentions, “[a]rray processors” had “been designed to use analog representations of numbers and analog circuits to perform computations. The SCAMP is such a machine.” ’201 Application, [0026]. However, analog computing was very primitive; for example, the ’201 Application notes that the “SCAMP” architecture “omits general division and multiplication from its design.” ’201 Application, [0026]. The impact of various implementation choices on the *precision* of an analog HDR execution unit was unpredictable, and

the specification fails to describe any correlation between an analog execution unit's implementation details and its precision.

91. The '201 Application does not describe what structural features differentiate silicon-implemented analog execution units that meet the claimed functional performance characteristics (X and Y) from those that do not, or how to adjust silicon-implemented analog execution units to meet the claimed imprecision characteristics. For instance, the '201 Application notes that the prior art "SCAMP" machines "provide low precision arithmetic, in which each operation *might* introduce *perhaps* an error of a few percentage points in its results." '201 Application, [0026]. But nowhere does the '201 Application explain how or when these error levels are achieved, or how they relate to any structure in any analog circuit. Moreover, the '201 Application never explains how to build an analog circuit that preforms operations in a way that produces any minimum error for any minimum percentage of inputs, as the claims of the '273 patent require.

92. The '201 Application identifies a prior-art analog design (the "SCAMP" design) alleged to have performed "a range of low precision" computations, but it only performed the computations on a "low dynamic range" of values (not HDR as claimed) and only for relatively simple operations like addition and subtraction (not multiplication or division). '201 Application, [0076] ("An example of an analog SIMD architecture is the SCAMP design (and related

designs) of Dudek. In that design values have low dynamic range, being accurate roughly to within 1%.”), [0077] (“Variations of the SCAMP design have been fabricated and used to perform a range of *low precision, low dynamic range* computations related to image processing. These designs *do not perform high dynamic range arithmetic*, nor do they include mechanisms for performing multiplication or division of values stored in Registers.”).

93. The ’201 Application states this prior art design “suggest[s] the general feasibility” of an analog implementation. ’201 Application, [0077] (“[T]he Dudek designs suggest the general feasibility of constructing *analog SIMD machines*. The following describes how to build *an analog SIMD machine that performs LPHDR arithmetic, and is thus an embodiment* of the present invention.”). However, the ’201 Application describes no modification alleged to enable the prior-art implementation to achieve an *analog-only* execution unit that could perform any high dynamic range mathematical operation as claimed, let alone do so with the degree of imprecision the claims require. The ’201 Application also does not explain how to modify the prior-art design in analog-only to perform the full range of operations the specification describes, including not only relatively simple multiplication operations the ’201 Application says the prior art analog system could not perform, but also far more complex trigonometric or exponentiation functions the ’273 patent says the claimed execution unit

encompasses. *See* '273 patent, 1:63-66, 11:44-48, 27:31-40, which I discuss in paragraph 64 above.

94. The '201 Application goes on to describe a *mixed* analog-digital floating-point implementation that uses an analog mantissa and digital exponent; but the application merely alleges the analog *representation* of the mantissa is “accurate roughly to *within* 1%.” '201 Application, [0076]-[0082]. As I discussed in paragraphs 92-93, the '201 Application discusses the prior art “SCAMP” design and says that it “suggest[s] the general feasibility” of an analog embodiment. '201 Application, [0077]. The application then purportedly “describes how to build an analog SIMD machine that performs LPHDR arithmetic,” which is “an embodiment of the present invention.” '201 Application, [0077]. This embodiment “*represents* values as a *mixture* of analog and digital forms.” '201 Application, [0078]. This embodiment “represents values as low precision, normalized, base 2 floating point numbers, where the *mantissa is an analog value* and the exponent is a binary digital value. The *analog value may be accurate to about 1%*, following the approach of Dudek, which is well within the range of reasonable analog processing techniques.” '201 Application, [0078]. In other words, the analog-represented inputs to the operation are *at most* 1% imprecise, but the '201 Application does not say whether any input reaches that maximum

imprecision or, if so, how many inputs reach that maximum imprecision, or, if so, how often (*e.g.*, given that analog values can unpredictably change).

95. Moreover, a POSA would have understood that having an imprecise input also does not disclose an execution unit as in the '273 patent's claims that introduces imprecision in its performance of a mathematical operation and *produces results* that differ from an exact mathematical calculation on the input signal by "at least $Y=0.05\%$ " for at least 5% of valid inputs as the claims recite. All the application says is that the *representation* of numbers is up to 1% inaccurate. But the '201 Application does not explain how this representation error relates (if at all) to the claimed output error, and a POSA would have understood that this representation error is not guaranteed to lead to an output error that meets the claims of the '273 patent.

96. Consider an example that imprecisely represents the decimal input 1.00 as 1.01, and the decimal input 2.00 as 1.98. Each of those inputs is 1% imprecise; however, a POSA would have understood that the *output* of multiplying those inputs could be perfectly precise—*e.g.*, $1.01 \times 1.98 = 1.9998$ —or could have some different percent error—*e.g.*, outputting 2.0 which is 0.01% (not 1%, and not "at least 0.05%" as claimed) different from the exact result of 1.9998.

97. The '201 Application also discloses "mean score error" results from running high-level software programs using a software approximation designed to

simulate the mixed analog/digital embodiment by simply *assuming* that it will produce some random amount of error somewhere between 0-1% in each operation. '201 Application, [0087] (“Applications are tested using two embodiments for the machine's arithmetic. One uses accurate floating point arithmetic but multiplies the result of each arithmetic operation by a uniformly chosen random number between 0.99 and 1.01. In the following discussion, this embodiment is denoted “fp+noise”. It may represent the results produced by an analog embodiment of the machine.”). However, that does not demonstrate that an actual mixed-analog hardware circuit would produce random error between 0-1%.

98. Also, as with the simulation of the logarithmic embodiment (which I discussed in Section V.B.2.a above), the “mean score error” for the high-level program is not the claimed error in an individual operation. *See* '201 Application, [0085]-[0126]. As I explained in Section V.B.2.a above with respect to the digital embodiment, the '201 Application describes a software “model” ([0085]) that simulates different embodiments of the alleged invention. *See* paragraph 84 above. One of the “two embodiments for the machine's arithmetic ... uses accurate floating point arithmetic but multiplies the result of each arithmetic operation by a uniformly chosen random number between 0.99 and 1.01 ... this embodiment is denoted ‘fp+noise’” and “may represent the results produced by an analog embodiment of the machine.” '201 Application, [0087]. The '201 Application

describes “mean score error” results for tests using this “fp+noise” embodiment in paragraph [0104]. However, for the same reasons I discussed in paragraph 87 above with respect to the “lms” embodiment, a POSA would have understood that the “mean score error” that the ’201 Application reports does **not** refer to error for individual arithmetic operations, or even individual weighted scores, but rather is the mean error for the **sum** of “over one million small positive values” that were themselves computed by another series of arithmetic operations. Nowhere in the discussion of its test results (*see* [0101]-[0110]) does the ’201 Application report error percentages for individual arithmetic operations, or explain how one can derive those percentages from the mean score errors reported. The ’201 Application also describes a test of an algorithm for “Removing motion blur in images” at paragraphs [0120]-[0126], and reports “mean pixel error” results for the “fp+noise” and “lms” embodiments (’201 Application, [0123]), but that discussion similarly lacks any disclosure of error percentages for individual arithmetic operations, or of how one can derive those percentages from the mean pixel errors reported.

99. Thus, a POSA would have understood that the ’201 Application does not describe **even one** species of an analog implementation (whether pure analog or mixed analog/digital) that falls within any challenged claim’s scope, let alone

support the full scope of the genus of analog implementations that the '273 patent says its claims cover.

100. Given the unpredictable nature of analog design using even conventional silicon-based technology, a POSA would not have known whether a particular analog implementation of an execution unit would possess the claimed functional performance characteristics without physically building and testing it. For corroborating evidence, *see, e.g.*, Hopper (Ex. 1059), 1:14-41 (explaining that “[w]ith analog circuitry ... transistors operate in the linear range. As a result, the behavior of the transistors is far more difficult to predict due to a host of variables, such as variations in process technology, the differences in the gain of the transistors, and noise for example. It is therefore difficult to model and simulate analog circuitry with a high degree of accuracy.”)

101. I noted above that the '201 Application describes a software simulation that “represent[s] the results produced by an analog embodiment of the machine” by “multipl[y]ing the result of each arithmetic operation by a uniformly chosen random number between .99 and 1.01.” *See* '201 Application, [0087] (“Applications are tested using two embodiments for the machine's arithmetic. One uses accurate floating point arithmetic but multiplies the result of each arithmetic operation by a uniformly chosen random number between 0.99 and 1.01. In the following discussion, this embodiment is denoted ‘fp+noise’. It may represent the

results produced by an analog embodiment of the machine.”). However, the ’201 Application’s assumption of this “represent[ation] of the *results* produced by an analog embodiment” (’201 Application, [0087]) does not tell a POSA how to actually *construct* an embodiment with these characteristics. Actually building a device that produced these “results” would have required trial and error, for the reasons I discussed in paragraph 100 above.

(2) The Digital Silicon Transistor-Based Implementation Is Not Representative of Execution Units Implemented Using Unpredictable and Nascent Technologies

102. The ’201 Application lists a number of nascent technologies and speculates that they “*may*” someday be used to implement non-silicon-based “execution units” that could perform some unspecified mathematical “operation” on digital or analog representations. *See* ’201 Application, [0139] (“embodiments of the present invention *may* be implemented using technologies that *may* enable other sorts of traditional digital and analog computing processors or other devices. Examples of such technologies include various nanomechanical and nanoelectronic technologies, chemistry based technologies such as for DNA computing, nanowire and nanotube based technologies, optical technologies, mechanical technologies, biological technologies, and other technologies...”).

103. But based on my review, the ’201 Application provides no description of how an execution unit could be designed using any one of the listed nascent

technologies (*e.g.*, nanomechanical or DNA computing), let alone a description of what implementation techniques would need to be used to ensure that any such execution unit would possess the challenged claims' low-precision functional performance characteristics.

104. Based on my knowledge and experience as someone of at least ordinary skill in the art, I am aware that the technologies listed in the '201 Application at [0139] were nascent and unpredictable in 2010. For example, nanotechnology involves "creation and use of structures, devices, and systems" of a size "at the level of atoms and molecules." *See, e.g.*, Ex. 1021, page 16 (publication by the National Nanotechnology Initiative corroborating the POSA's understanding that "nanotechnology ... comprises the following three factors: 1. Research and technology development at the atomic, molecular, or macromolecular levels ... 2. Creation and use of structures, devices, and systems that have novel properties and functions because of their small and/or intermediate size, at the level of atoms and molecules; [and] 3. Ability for atomic-scale control or manipulation.").

105. Around the time of the '201 Application's alleged invention, POSAs viewed nanotechnology as a futuristic manufacturing technology that might *someday* allow for more complex and precise structures; however, no nanotechnology computer or execution unit of the type the '273 patent claims had

been realized. As a 2006 article regarding nanotechnology explained, “the majority of nanoscale scientists and engineers believe it is too early to try to predict the ultimate capabilities of such systems.” Ex. 1021, page 107. The article described nanotech computing as “visionary engineering,” and contrasted it with “conventional engineering” that was “concerned with the design of things that can be built more or less immediately.” *Id.* As the article put it succinctly: “[S]howing that [a structure] can, in principle, exist[] does not tell one how to build it, and these arguments do not yet constitute a research strategy or a research plan.” *Id.* This is consistent with my knowledge of how POSAs viewed, *e.g.*, nanomechanical and nanoelectronic computing technologies in the 2010 timeframe.

106. Designing computer circuits using nanotechnology was (and remains) a nascent and unpredictable technology. For evidence corroborating that this was the POSA’s understanding, *see, e.g.*:

- Ex. 1047 (published 2010), 36 (“one form of nanoscale computer architecture” is “nano-mechanical computing devices”), 36-37 (“In order to ***produce such nanoscale architectures***, molecular manufacturing and mechanosynthesis ***techniques must be understood***, including the use of hydrocarbon assemblers. As ***this technology is not yet realized***, the use of a variety of research tools to model and simulate the proposed architecture is required.”);
- Ex. 1048 (published 2019), 3 (“While carbon nanotubes have a lot of potential, manufacturing them into transistors is a real challenge ... Currently, the material used in these chips requires 99.999999 percent purity, which is virtually impossible.”);

107. The '201 Application suggests that nanotechnology for computing was not yet “reach[ed]” when it predicts that the advantages of its execution unit when “fabricated with current state of the art technology” would “persist as fabrication technology continues to improve, even as we *reach* nanotechnology or other implementations for digital and analog computing.” '201 Application, [0128] (“a digital embodiment of the present invention built as a large silicon chip fabricated with current state of the art technology might perform tens of thousand of arithmetic operations per cycle, as opposed to hundreds in a conventional GPU or a handful in a conventional multicore CPU. These ratios reflect an architectural advantage of embodiments of the present invention that *should* persist as fabrication technology *continues to improve, even as we reach nanotechnology* or other implementations for digital and analog computing.”). Despite this, the '201 Application provides no details concerning nanotechnology-based circuits that would suggest to the POSA that the inventor possessed a nanotechnology-based execution unit that could perform *any* LPHDR operation in 2010, let alone the full scope of operations (including non-linear and trigonometric functions) that the specification mentions, with the degree of imprecision specified in the '273 patent's claims. See '201 Application, [0006] (mentioning that “a silicon MOSFET transistor is a device capable of performing interesting non-linear

operations, such as exponentiation,” but providing no explanation of how to implement that operation, and including no mention of non-silicon or non-transistor technologies), [0059] (stating that “LPHDR operations other than and/or in addition to addition and multiplication may be supported,” but providing no explanation of how to implement those operations on any technology), [0144] (referring to “an embodiment in which LPHDR arithmetic elements can perform one or more operations (perhaps including, for example, trigonometric functions),” but providing no explanation of how to implement those operations on any technology).

108. In addition to nanotechnology, numerous other technologies listed in the '201 Application were nascent and unpredictable, and a POSA would not have known how to implement the claimed execution unit using them. For evidence corroborating that this was how POSAs viewed these technologies, *see, e.g.*,

- Ex. 1049 (published 2015), 1 (“For more than 20 years, researchers have explored how ***DNA*** could be used as a material for computing. It ***sounds promising*** because of the incredible density of data in DNA ... But using DNA in this way is ***untenably slow*** for the kinds of jobs we expect computers to do.”);
- Ex. 1050 (published 2013), 3 (“Of all the emerging materials and new ideas held up as possible saviors—nanowires, spintronics, graphene, ***biological*** computers—no one has made a central processing unit based on any of them.”);
- Ex. 1051 (published 2017), 503 (“***Optical computing*** has been an active topic of ***research*** for over seven decades, although ***solutions*** have been elusive.”).

109. In light of the state of these technologies in 2010, a POSA reading the '201 Application would have recognized that it aspirationally lists a number of technologies that “may” someday be capable of implementing an HDR “execution unit” with the '273 patent’s claimed functional performance (imprecision) characteristics, but would not have understood the '201 Application to demonstrate that its inventor actually possessed any such “execution unit” given the absence of any related description.

(3) Conclusion: The Digital Silicon Transistor-Based Implementation Is Not Representative of the Claimed Genus

110. Given the vast number of aspirational species covered by the '273 patent’s claimed genus of all HDR execution units (no matter how implemented) that operate with the claimed amount of imprecision, including numerous subgenera that the specification suggests are covered but does not actually describe, even if the Board were to find that the '201 Application describes a silicon-implemented transistor-based digital embodiment meeting the claims (it does not, as I explained in Section V.B.2.a above), a POSA would have understood that that species is not remotely representative of the entire genus covered by the claims.

c. The '201 Application Does Not Disclose Structural Features Common to the Claimed Genus

111. As I explained in Section V.B.2 above, all of the '273 patent's independent claims recite an LPHDR execution unit with certain functional performance characteristics. The '201 Application does not describe the claimed genus of LPHDR execution units using *any* structural features. Instead, it characterizes the genus only using functional performance characteristics that specify the precision with which the execution unit operates; any execution unit that operates with the specified degree of imprecision is included in the genus regardless of the structural features used to implement the execution unit.

112. As I explained in Section V.B.1 above, the genus claimed by the challenged claims of the '273 patent encompasses digital and analog implementations that may be implemented using technologies ranging from traditional silicon transistors to nanomechanical or DNA computers whose “structural features” are undisclosed. I find, and in my opinion a POSA would have found, nothing in the '201 Application suggesting that there are common structural features among, *e.g.*, a traditional silicon-based circuit that processes digital representations and nanomechanical and DNA-based execution units that operate on analog number representations.

113. This is not surprising. As a POSA reading the language of the challenged claims would have understood, there are no such common structural

features because the genus is not defined by anything structural; rather, the only thing common to the genus is that all execution units within the genus share a functional performance characteristic, *i.e.*, they operate with the specified degree of minimum imprecision.

C. Enablement: The '201 Application Does Not Enable the Full Scope of Any Challenged Claim

1. Breadth of the Claims and Nature of the Invention

114. As I explained in Section V.B.1 above, the challenged claims of the '273 patent recite an “execution unit” that can be virtually *anything* “adapted to execute” *any* mathematical “operation” on an input signal that represents a high-dynamic-range numerical value (via digital or analog representation) so long as the execution unit possesses the claimed low-precision functional performance characteristics. *See*, limitations [1B1]-[1B2], [33B1]-[33B2], [36B1]-[36B2], [68B1]-[68B2]. Those imprecision performance characteristics are also recited with no upper limit. *See e.g.*, Claim 1 (“for *at least* X=5%” and “differs by *at least* Y=0.05%”).

115. The nature of the alleged invention is an “execution unit” that can perform arithmetic operations on numerical values of high dynamic range with low precision. The '273 patent states:

Embodiments of the present invention are directed to a processor or other device, such as a programmable and/or massively parallel

processor or other device, which includes processing elements designed to perform arithmetic operations (possibly but not necessarily including, for example, one or more of addition, multiplication, subtraction, and division) on numerical values of low precision but high dynamic range (“LPHDR arithmetic”). Such a processor or other device may, for example, be implemented on a single chip. Whether or not implemented on a single chip, the number of LPHDR arithmetic elements in the processor or other device in certain embodiments of the present invention significantly exceeds (e.g., by at least 20 more than three times) the number of arithmetic elements in the processor or other device which are designed to perform high dynamic range arithmetic of traditional precision (such as 32 bit or 64 bit floating point arithmetic). In some embodiments, “low precision” processing elements perform arithmetic operations which produce results that frequently differ from exact results by at least 0.1% (one tenth of one percent). This is far worse precision than the widely used IEEE 754 single precision floating point standard. Programmable embodiments of the present invention may be programmed with algorithms that function adequately despite these unusually large relative errors. In some embodiments, the processing elements have “high dynamic range” in the sense that they are capable of operating on inputs and/or producing outputs spanning a range at least as large as from one millionth to one million.

’273 patent, 2:11-39. The claimed invention is defined not by any structural aspect of an “execution unit,” but solely by its performance characteristics. *See my discussion in Section V.B.1.b above.*

2. State of the Art, Level of POSA's Skill, Level of Unpredictability in the Art, and Quantity of Experimentation Needed to Practice the Alleged Invention Based on the Disclosure

116. As I explained in Section V.B.1.a above, the '201 Application's laundry list of potential implementations makes clear that the claimed "execution unit" covers, among other things, nanotechnology-based or DNA-based "circuits," operating on analog representations of numbers. And as I discussed in Section V.B.2 above, these technologies were nascent and were entirely unpredictable. It would not have been within the POSA's level of ordinary skill, without undue experimentation, to develop an HDR execution unit that could perform *any* mathematical operation using these aspirational technologies, let alone one that could perform complex non-linear or trigonometric functions of the type the specification says are covered by the claims.

117. Compounding the complexity, the challenged claims cover analog representations using a wide range of physical characteristics ("charges, currents, voltages, frequencies, pulse widths, pulse densities, various forms of spikes") including "other forms" that the specification says are possible but nowhere identifies, and the challenged claims cover non-deterministic implementations that generate different results at different times for the same input. *See* my discussion of the claims' scope in Section V.B above. Even a more conventional implementation that uses a silicon circuit to operate on analog number

representations was unpredictable, as I explain in Section V.B.2.b(1) above. POSAs knew how to develop *some* silicon circuits that operated on analog numerical representations, but the patent's specification never explains what implementation decisions would purportedly result in a circuit possessing the claimed imprecision characteristics (X and Y percentages) versus a circuit that would not. The '201 Application only states that the existence of prior-art analog silicon circuits "suggest[s] the general feasibility" of creating an "execution unit" with the claimed imprecision characteristics ('201 Application, [0077]), but does not inform a POSA how to actually implement an analog execution unit possessing those characteristics.

118. To arrive at such an "execution unit," a POSA would have needed to undertake significant experimentation. For example, a POSA would have had to measure electrical characteristics such as charge injection, voltage coupling, and parasitic capacitances to determine whether the required error could be achieved. A POSA would have understood that these characteristics are heavily dependent on physical properties of silicon circuits that are not easily adjusted. In my opinion, arriving at an analog circuit within the claims, given the application's failure to provide any working examples or otherwise provide POSAs with direction, would have required a significant amount of experimentation especially given the

unpredictable nature of analog technology. In my opinion, such experimentation necessary to achieve the claimed invention would have be undue and unreasonable.

119. The challenged claims encompass subject matter that was (and remains) only theoretically plausible. As I discuss in Section V.B.2.b(2) above, the '201 Application recites a long list of technologies that “may” be used as an alternative to silicon to implement LPHDR execution units, and all of these technologies were still in experimental stages even years after the '201 Application was filed. The '273 patent recites this same list. *See* '273 patent, 26:17-31. And the challenged independent claims contain no language that would exclude execution units made from these technologies (and indeed do not mention any particular implementation technology at all).

3. Amount of Direction Provided, Including Working Examples, in the Specification

120. In my opinion, a POSA reading the '201 Application would have understood that it lacks direction or working examples for most of the implementations covered by the claims as to meeting the recited functional performance characteristics (X and Y).

121. As I discussed in Section V.B.2 above, for the vast majority of the many varieties of execution unit implementations that the '273 patent's claims purport to cover, there is no example whatsoever in the '201 Application of such an implementation, nor any hint of how to accomplish such an implementation.

Even if the '201 Application were considered to describe digital floating-point and logarithmic (LNS) implementations using conventional silicon transistor-based circuits, it provides no guidance or working examples of the numerous other implementations of an “execution unit” that the specification says the alleged invention encompasses (such as, for example, DNA-based execution units or nanomechanical execution units).

122. Instead, the application *lists* numerous futuristic technologies and alleges they “may enable” analog as well as digital “computing processors or other [unspecified] devices” ('201 Application, [0139]) but never explains how to make an “execution unit” as claimed using an analog implementation with these futuristic technologies. A POSA would not have been able to implement the full scope of what the specification alleges the claims cover given the little (to no) direction that the specification provides.

D. Bates-2010 Renders Obvious Claims 1-70

1. Bates-2010's Status As Prior Art

123. I am informed and understand that Bates-2010 (Ex. 1006) is the official publication by the U.S. Patent and Trademark Office of the '201 Application, published on December 23, 2010. *See* Bates-2010, page 1, item (21) (“Appl. No.” is “12/816,201”), item (43) (“Pub. Date” is “Dec. 23, 2010”). The text of Bates-2010 is identical to the text of the '201 Application, but some of the

paragraphs in the two applications are numbered differently. The paragraph numbers match until number [0104], but after this, they diverge; for example, the paragraph numbered [0105] in the '201 Application is numbered [0119] in Bates-2010. This appears to be because certain lines of text were assigned separate paragraph numbers in Bates-2010 that were not assigned separate paragraph numbers in the '201 Application. *Compare* '201 Application, [0104]-[0105] *with* Bates-2010, [0104]-[0119].

124. I am informed and understand that because the '201 Application does not provide sufficient written description support and enablement for the challenged claims of the '273 patent, those claims are not entitled to claim priority to the '201 Application's filing date, and Bates-2010 is prior art to the challenged claims of the '273 patent.

2. An Obvious Implementation of Bates-2010's Silicon Transistor-Based Logarithmic Execution Unit

125. Bates-2010 discloses a silicon-implemented execution unit that operates on digital representations of numbers using transistors. Bates-2010, [0035], [0061], FIG. 4. Bates-2010 says that such an implementation can use "floating point arithmetic" with "binary mantissas of no more than 10 bits plus a sign bit and binary exponents of at least 5 bits plus a sign bit" to represent values "with a precision of about 0.1%." Bates-2010, [0035] ("One variety of LPHDR arithmetic represents values from one millionth up to one million with a precision

of about 0.1%. If these values were represented and manipulated using the methods of floating point arithmetic, they would have *binary* mantissas of no more than 10 bits plus a sign bit and binary exponents of at least 5 bits plus a sign bit.”), [0162] (“[E]mbodiments of the present invention may represent values in any of a variety of ways, such as by using digital ... representations, such as ... floating point representations”). In this implementation, the execution unit operates on numbers with dynamic range “from one millionth up to one million” to perform operations including, e.g., “multiply[ing].” Bates-2010, [0035].

126. Bates-2010 does not identify a minimum relative error (Y) produced by the above-described floating-point execution unit for a minimum percentage (X) of inputs, but Bates-2010 elsewhere suggests that an execution unit (divorced from any specific implementation) can be implemented to produce relative error (Y) of at least 0.05% for at least 5% (X) of possible valid inputs. Bates-2010 states:

The frequency with which LPHDR arithmetic elements may yield only approximations to correct results may vary from implementation to implementation. For example, consider an embodiment in which LPHDR arithmetic elements can perform one or more operations (perhaps including, for example, trigonometric functions) In such an example embodiment, consider further a fraction F of the valid inputs and a relative error amount E by which the result calculated by an LPHDR element may differ from the mathematically correct result.

In certain embodiments of the present invention, for each LPHDR arithmetic element, for at least one operation that the LPHDR unit is capable of performing, for at least fraction F of the possible valid inputs to that operation, for at least one output signal produced by that operation, the statistical mean, over repeated execution, of the numerical values represented by that output signal of the LPHDR unit, when executing that operation on each of those respective inputs, differs by at least E from the result of an exact mathematical calculation of the operation on those same input values, where F is 1% and E is 0.05%. In several other example embodiments, F is not 1% but instead is one of 2%, or 5%, or 10%, or 20%, or 50%. For each of these example embodiments, each with some specific value for F, there are other example embodiments in which E is not 0.05% but instead is 0.1%, or 0.2%, or 0.5%, or 1%, or 2%, or 5%, or 10%, or 20%.

Bates-2010, [0172]. A POSA would have understood that “F” and “E” in the passage above correspond to “X” and “Y,” respectively, in the claims of the ’273 patent, as I show below.

Bates-2010, [0172]	’273 patent, Claim 1
... consider further a fraction F of the valid inputs and a relative error amount E by which the result calculated by an LPHDR element may differ from the mathematically correct result. In certain embodiments of the present invention, for each LPHDR arithmetic element, for at least one operation that the LPHDR unit is capable of	[1B2] for at least <i>X=5% of the possible valid inputs to the first operation</i> , the statistical mean, over repeated execution of the first operation on each specific input from the at least X % of the possible valid inputs to the first operation, of the numerical values represented by the first output signal of the LPHDR unit executing the first

<p>performing, for at least <i>fraction F of the possible valid inputs to that operation</i>, for at least one output signal produced by that operation, the statistical mean, over repeated execution, of the numerical values represented by that output signal of the LPHDR unit, when executing that operation on each of those respective inputs, <i>differs by at least E from the result of an exact mathematical calculation of the operation on those same input values</i>, where F is 1% and E is 0.05% ...</p>	<p>operation on that input <i>differs by at least $Y=0.05\%$ from the result of an exact mathematical calculation of the first operation on the numerical values of that same input</i>; and</p>
--	--

127. Given this suggestion in Bates-2010 ([0172]), a POSA would have understood how to emulate in software the performance of the floating-point execution unit, performing a particular type of operation (*e.g.*, multiplication), to determine whether the disclosed operation that “represent[s] and manipulate[s]” floating-point numbers with 10-bit mantissas (Bates-2010, [0035]) would produce this level of imprecision that Bates-2010 ([0172]) suggests that execution units can achieve.

128. Based on the software emulation, a POSA would have known how to decrease the number of mantissa bits utilized until the relative error amounts for the fractions of valid inputs suggested in paragraph [0172] of Bates-2010 are met, and would have had a reasonable expectation of success in doing so. For corroborating evidence, *see*, for example, Dockser (Ex. 1007), which explains that

a POSA would have known that floating-point “precision ... is defined by the number of bits used to represent the mantissa[;] [t]he more bits in the mantissa, the greater the precision.” Dockser, [0002]. *See also* Tong (Ex. 1008), 278 (“It is obvious that power dissipation in an FP unit can be reduced by using fewer bits in the FP representation. However, fewer bits reduces precision and might result in a less accurate output.”).

129. As I demonstrate below, the silicon-implemented transistor-based digital floating-point execution unit disclosed in Bates-2010’s paragraph [0035], modified as needed to achieve the performance characteristics that Bates-2010’s paragraph [0172] suggests achieving, and implemented in a computing device in accordance with Bates-2010’s suggestion (*e.g.*, at [0007]), meets each element of the challenged claims of the ’273 patent.

3. Bates-2010 Renders Obvious Claim 1 of the ’273 patent

a. [1A1] A device comprising: at least one first low precision high dynamic range (LPHDR) execution unit

130. Bates-2010 discloses that “[e]mbodiments of the present invention are directed to a processor or other device ...which includes processing elements designed to perform arithmetic operations ... on numerical values of low precision but high dynamic range (‘LPHDR arithmetic’).” Bates-2010, [0007]. It also

explicitly claims “devices” including one or more LPHDR execution units. Bates-2010, Claims 1-2.

131. An exemplary device is depicted in Figure 1, which Bates-2010 states is “an example overall design of a SIMD processor according to one embodiment of the present invention.” Bates-2010, [0009].

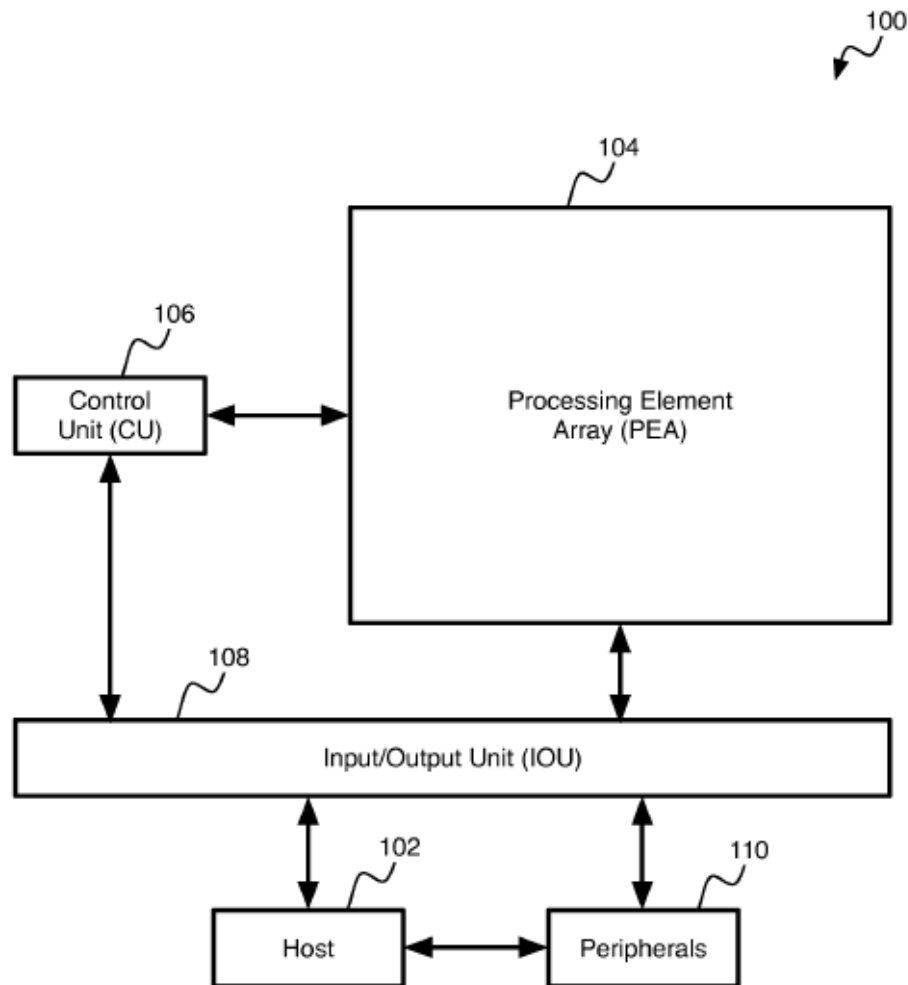


FIG. 1

132. Figure 1's SIMD processor is a "device" as claim 1 (limitation [1A1]) of the '273 patent recites. *See, e.g.*, Bates-2010, [0006] ("Real computers are built as physical devices...."), [0040] ("Various computing devices implemented according to embodiments of the present invention will now be described. Some of these embodiments may be an instance of a SIMD computer architecture."), [0157] (referring to "conventional computing devices"), [0173] ("For certain devices (such as computers or processors or other devices) embodied according the present invention, the number of LPHDR arithmetic elements in the device (e.g., computer or processor or other device)"), [0175] ("Embodiments of the present invention may, however, be implemented in devices in addition to or other than processors. For example, a computer including a processor and other components (such as memory coupled to the processor by a data path), wherein the processor includes components for performing LPHDR operations in any of the ways disclosed herein, is an example of an embodiment of the present invention. More generally, any device or combination of devices, whether or not falling within the meaning of a "processor," which performs the functions disclosed herein may constitute an example of an embodiment of the present invention.")

133. This SIMD processor device includes a control unit 106 and "a collection of many processing elements (PEs) 104." Bates-2010, [0043]. Bates-2010 explains that a processing element (PE) is one example of a "kind of

execution unit” that is an “LPHDR element[,]” – in other words, the PE is an example of an LPHDR execution unit. Bates-2010, [0042] (“references herein to ‘processing elements’ within embodiments of the present invention should be understood more generally as any kind of execution unit, whether for performing LPHDR operations or otherwise”), [0043] (“An example SIMD computing system 100 is illustrated in FIG. 1. The computing system 100 includes a collection of many processing elements (PEs) 104. ... One embodiment of the present invention is a SIMD computing system of the kind shown in FIG. 1, *in which one or more (e.g., all) of the PEs in the PEA 104 are LPHDR elements*, as that term is used herein.”).

134. Bates-2010 thus expressly suggests implementing a “collection” of execution units, including Bates-2010’s silicon-implemented transistor-based digital floating-point execution units, in a device such as a SIMD processor.

- b. [1A2] adapted to execute a first operation on a first input signal representing a first numerical value to produce a first output signal representing a second numerical value,**

135. Bates-2010 discloses that the floating-point execution unit performs operations such as “multiply[ing]” on floating-point representations of numerical “values.” Bates-2010, [0035] (“One variety of LPHDR arithmetic represents values from one millionth up to one million with a precision of about 0.1%. If these values were represented and manipulated using the methods of floating point

arithmetic, they would have binary mantissas of no more than 10 bits plus a sign bit and binary exponents of at least 5 bits plus a sign bit. However, the *circuits to multiply and divide these floating point values* would be relatively large.”). A POSA would have understood that performing multiplication on an input representation of a first numerical value results in the production of an output signal representing a second numerical value (the product of the multiplication).

136. Bates-2010 depicts “an example design for a Processing Element” in FIG. 4, where a “processing element[]” (PE) is one example of a “kind of execution unit.” Bates-2010, [0012] (“FIG. 4 is an example design for a Processing Element according to one embodiment of the present invention.”), [0042] (“references herein to ‘processing elements’ within embodiments of the present invention should be understood more generally as any kind of execution unit, whether for performing LPHDR operations or otherwise”).

137. The PE execution unit receives “input” and produces “output” that both “take the form of electrical signals representing numerical values.” Bates-2010, [0055] (“The *input, output*, and intermediate ‘values’ *received by, output by*, and operated on *by the PE 400* may, for example, *take the form of electrical signals representing numerical values.*”).

138. The input signals provide “local data” on which the PE “operate[s]” to produce “results” providing the output signals. Bates-2010, [0045] (“the PEs...

perform computations... on data stored locally in each PE”), [0048] (Input/Output Unit 108 “get[s] data into and out of” PEs), [0051] (“The design [shown in Figure 3] may permit data to be pushed from the [input/output unit] 108 inward to any PE...”), [0055] (“Each PE needs to operate on its local data.”), [0057] (“control signals ... specify ... which operations should be performed by the Logic 406 or Arithmetic 408 or other processing mechanisms [and] where the results should be stored in the Registers ...”).

139. The “LPHDR operations” performed by a PE include “multiplication.” Bates-2010, [0059] (“LPHDR operations other than and/or in addition to addition and multiplication may be supported.”).

140. A POSA would thus have understood that each PE (an example generic execution unit in Bates-2010) within Figure 1’s SIMD processor is “adapted to execute a first operation,” *e.g.*, multiplication, “on a first input signal representing a first numerical value to produce a first output signal representing a second numerical value,” as claim 1 (limitation [1A2]) of the ’273 patent recites.

141. Claims 1 and 2 of Bates-2010 further confirm this implementation, and claim 2 of Bates-2010 repeats limitation [1A2] of the ’273 patent verbatim. *See* Bates-2010, claim 1 (“adapted to execute, in parallel, a first plurality of operations on a first plurality of input signals representing a first plurality of numerical values to produce a first plurality of output signals representing a second

plurality of numerical values”), claim 2 (“adapted to execute a first operation on a first input signal representing a first numerical value to produce a first output signal representing a second numerical value”).

142. Bates-2010 also depicts in Figure 6 “an example digital implementation of [an] LPHDR arithmetic unit,” which is a component of a PE that “performs LPHDR arithmetic operations” including “LPHDR multiplication.” Bates-2010, [0014] (“FIG. 6 is an example design for an LPHDR arithmetic unit according to one embodiment of the present invention.”), [0055] (“*The LPHDR arithmetic unit 408 performs LPHDR arithmetic operations*, as that term is used herein. The input, output, and intermediate ‘values’ received by, output by, and operated on by the PE 400 may, for example, take the form of electrical signals representing numerical values.”), [0067] (“FIG. 6 shows an example digital implementation of the LPHDR arithmetic unit 408 for the representation illustrated in FIG. 5.”).

143. The LPHDR arithmetic unit 408 receives “inputs, A 602a and B 602b, and produces an output 602c,” which “take the form of electrical signals representing numerical values.” Bates-2010, [0067] (“FIG. 6 shows an example digital implementation of the LPHDR arithmetic unit 408 for the representation illustrated in FIG. 5. The unit 408 receives two inputs, A 602a and B 602b, and produces an output 602c. *The inputs 602 a-b and output 602c may, for example,*

take the form of electrical signals representing numerical values according to the representation illustrated in FIG. 5, as is also true of signals transmitted within the unit 408 by components of the unit 408.”).

144. A POSA would thus have understood that the “LPHDR execution unit” recited in limitation [1A2] of the ’273 patent is broad enough to cover either or both of (1) PE 400 and (2) LPHDR arithmetic unit 408 within PE 400 in Bates-2010, both of which meet the limitations recited in limitations [1A1]-[1A2].

c. [1B1] wherein the dynamic range of the possible valid inputs to the first operation is at least as wide as from 1/65,000 through 65,000

145. Bates-2010 discloses that various “[e]mbodiments” of “LPHDR arithmetic mechanisms” can be included in a PE. Bates-2010, [0060] (“One aspect of embodiments of the present invention that is unique is the inclusion of **LPHDR arithmetic mechanisms** in the PEs. *Embodiments of such mechanisms* will now be described.”).

146. One of these embodiments is the above-discussed floating-point embodiment that “represents values from *one millionth up to one million*,” which is “at least as wide as” (and indeed wider than) “from 1/65,000 through 65,000,” as claim 1 (limitation [1B1]) of the ’273 patent recites. [0035] (“One variety of LPHDR arithmetic represents values from one millionth up to one million with a precision of about 0.1%.”).

147. Bates-2010 also explicitly suggests operating on the specific dynamic range recited in limitation [1B1] of the '273 patent. Bates-2010, [0171] (“As yet another example, in certain embodiments, a LPHDR arithmetic element processes values in a space which may range approximately from one sixty five thousandth to sixty five thousand.”). Additionally, Bates-2010’s claims 1 and 2 provide verbatim disclosure of limitation [1B1] of the '273 patent. Bates-2010, claim 1 (“the dynamic range of the possible valid inputs to the at least one operation is at least as wide as from 1/65,000 through 65,000”), claim 2 (“wherein the dynamic range of the possible valid inputs to the first operation is at least as wide as from 1/65,000 through 65,000”).

148. A POSA would have understood that implementing each execution unit (*e.g.*, PE) within Figure 1’s SIMD processor to perform a first operation, *e.g.*, multiplication, on a range of possible valid inputs “at least as wide as from 1/65,000 through 65,000,” as recited in limitation [1B1] of the '273 patent, would have been a straightforward implementation given Bates-2010’s explicit suggestion to do so.

- d. **[1B2] for at least X=5% of the possible valid inputs to the first operation, the statistical mean, over repeated execution of the first operation on each specific input from the at least X % of the possible valid inputs to the first operation, of the numerical values represented by the first output signal of the LPHDR unit executing the first operation on that input differs by at least Y=0.05% from the result of an exact**

mathematical calculation of the first operation on the numerical values of that same input; and

149. Bates-2010 discloses that:

[F]or each LPHDR arithmetic element, for at least one operation that the LPHDR unit is capable of performing, for at least fraction F of the possible valid inputs to that operation, for at least one output signal produced by that operation, the statistical mean, over repeated execution, of the numerical values represented by that output signal of the LPHDR unit, when executing that operation on each of those respective inputs, differs by at least E from the result of an exact mathematical calculation of the operation on those same input values

Bates-2010, [0172]. Variable “F” in Bates-2010 corresponds to variable “X” in limitation [1B2], and variable “E” in Bates-2010 corresponds to variable “Y” in limitation [1B2], as I explained in paragraph 126 above.

150. Bates-2010 explicitly suggests implementing an execution unit to produce a relative error amount “E” (“Y” as claimed) of at least 0.05% for a fraction of the possible valid inputs “F” (“X” as claimed) of at least 5%, where the relative error amount is determined by measuring how much “the statistical mean, over repeated execution, of the numerical values represented by that output signal of the LPHDR unit” differs from “the result of an exact mathematical calculation of the operation on those same input values.” Bates-2010, [0172] (“In certain embodiments of the present invention,... F is 1% and ***E is 0.05%***. In several other

example embodiments, *F* is not 1% but instead is one of 2%, or 5%, or 10%, or 20%, or 50%. For each of these example embodiments, each with some specific value for *F*, there are other example embodiments in which *E* is not 0.05% but instead is 0.1%, or 0.2%, or 0.5%, or 1%, or 2%, or 5%, or 10%, or 20%.”). It would have been obvious to a POSA to implement Bates-2010’s floating-point execution unit as a deterministic unit (as digital circuits conventionally are), such that the claimed “statistical mean, over repeated execution of the first operation on each specific input,” is the same as the output of any single execution of that operation on that input, regardless of how many times the execution is repeated. For evidence corroborating that this was background knowledge for a POSA, *see, e.g.,* Weiss (Ex. 1011), 1:40-42 (“conventional digital circuits are traditionally deterministic.”).

151. As I explained in Section V.D.2 above (*e.g.*, paragraph 128), a POSA would have understood how to implement Bates-2010’s floating-point execution unit to achieve the functional performance characteristics suggested in [0172], which would have resulted in the execution unit meeting limitation [1B2] of the ’273 patent.

152. For example, for a multiplication operation performed on floating-point input values with 10-bit mantissas as described in Bates-2010 at [0035], a POSA would have known how to write a computer program to compute, for every

possible valid pair of floating-point input values with 10-bit mantissas: (1) the claimed “result of an exact mathematical calculation” of the product of those input values, which is a product including up to a 20-bit mantissa, and (2) the relative error by which the output (when represented using only a 10-bit mantissa as Bates-2010 suggests [0035]) differs from the exact mathematical calculation. As I explain in Appendix I.B and I.D below, a POSA would have known how to write such a program using programming tools that were available to a POSA by 2010; *e.g.*, the C programming language and Nvidia’s “CUDA” toolkit for running software on Nvidia graphics processing units (GPUs).

153. If the POSA’s computer program demonstrated that the floating-point execution unit using 10-bit-mantissas disclosed in Bates-2010 ([0035]) did not meet the relative error E of at least 0.05% for a fraction F of the possible valid inputs of at least 5% as suggested by Bates-2010 in [0172], then the POSA would have known that, in order to achieve those desired E and F values, the number of mantissa bits should be reduced. A POSA would have known how to reduce the number of mantissa bits and re-run the computer program until a reduced number of mantissa bits was arrived at that yielded the desired E and F values suggested in [0172]. This would have been a simple matter of modifying the code to use a smaller number of mantissa bits, re-running the test, and then repeating as needed

until the number of mantissa bits is sufficiently small that the unit produces error meeting the desired E and F values.

154. After the POSA determined a number of mantissa bits that produced the desired E (at least 0.05%) and F (at least 5%) values (from Bates-2010, [0172]) using the computer program, a POSA would have implemented an execution unit using that number of mantissa bits, and such an execution unit would have met limitation [1B2] of the '273 patent.

4. Claims 3, 5, 7-8, 9-10

155. Claims 3, 7, and 9 of the '273 patent recite that “the at least one first LPHDR execution unit” of claim 1 “comprises at least” 10, 100, and 500 “LPHDR execution units,” respectively. Claims 5, 8, and 10 recite that the number of LPHDR execution units in the device of claim 1 exceeds “the non-negative integer number of execution units...adapted to execute at least the operation of multiplication on floating point numbers that are at least 32 bits wide” by at least 10, 100, and 500, respectively.

Claim 3
3. The device of claim 1, wherein the at least one first LPHDR execution unit comprises at least ten LPHDR execution units.
Claim 5
5. The device of claim 1, wherein the number of LPHDR execution units in the device exceeds by at least ten the non-negative integer number of execution units in the device adapted to execute at least the operation of multiplication on floating point numbers that are at least 32 bits wide.

Claim 7
7. The device of claim 1, wherein the at least one first LPHDR execution unit comprises at least one hundred LPHDR execution units.
Claim 8
8. The device of claim 1, wherein the number of LPHDR execution units in the device exceeds by at least one hundred the non-negative integer number of execution units in the device adapted to execute at least the operation of multiplication on floating point numbers that are at least 32 bits wide.
Claim 9
9. The device of claim 1, wherein the at least one first LPHDR execution unit comprises at least five hundred LPHDR execution units.
Claim 10
10. The device of claim 1, wherein the number of LPHDR execution units in the device exceeds by at least five hundred the non-negative integer number of execution units in the device adapted to execute at least the operation of multiplication on floating point numbers that are at least 32 bits wide.

156. Bates-2010 discloses that Figure 1's SIMD processor includes a "collection of many" execution units. Bates-2010, [0009], ("FIG. 1 is an example overall design of a SIMD processor according to one embodiment of the present invention."), [0043] ("An example SIMD computing system 100 is illustrated in FIG. 1. The computing system 100 includes a collection of many processing elements (PEs) 104."). Bates-2010 further discloses that "[f]or certain devices (such as computers or processors or other devices) embodied according to the present invention, the number of LPHDR arithmetic elements in the device...exceeds the number, possibly zero, of arithmetic elements in the device which are designed to perform high dynamic range arithmetic of traditional

precision (that is, floating point arithmetic with a word length of 32 or more bits).”
Bates-2010, [0173].

157. Bates-2010 also states that “in certain embodiments,” the number of LPHDR arithmetic elements exceeds the number of traditional-precision arithmetic elements by over “one hundred,” and even by over “five thousand.” *See* Bates-2010, [0173] (“If N_L is the total number of LPHDR elements in such a device, and N_H is the total number of elements in the device which are designed to perform high dynamic range arithmetic of traditional precision, then N_L exceeds $T(N_H)$, where T is some function. Any of a variety of functions may be used as the function T . For example, ... the number of LPHDR arithmetic elements in the device may exceed one hundred more than five times the number of arithmetic elements in the device, if any, designed to perform high dynamic range arithmetic of traditional precision. As yet another example, in certain embodiments, the number of LPHDR arithmetic elements in the device may exceed one thousand more than five times the number of arithmetic elements in the device, if any, designed to perform high dynamic range arithmetic of traditional precision. As yet another example, in certain embodiments, the number of LPHDR arithmetic elements in the device may exceed five thousand more than five times the number of arithmetic elements in the device, if any, designed to perform high dynamic range arithmetic of traditional precision.”).

158. A POSA would have understood Bates-2010's above suggestions to implement Figure 1's SIMD processor (i.e., the claimed "device" of the '273 patent's claim 1) to include up to over five thousand more LPHDR execution units than other execution units to apply to all of Bates-2010's disclosed execution unit embodiments, including the silicon transistor-based digital floating-point embodiment. Alternatively, it would have been obvious to a POSA to implement a SIMD processor that adopts both Bates-2010's suggestion to include up to over 5,000 more LPHDR execution units, and Bates-2010's suggestion to implement the LPHDR execution units as silicon-implemented transistor-based digital floating-point execution units. This obvious implementation of Bates-2010's device, following Bates-2010's express suggestion to include up to over "five thousand" more LPHDR execution units (implemented as silicon-implemented transistor-based digital floating-point execution units) than execution units adapted to execute multiplication on 32-bit-wide floating-point numbers, comprises "at least" 10, 100, and 500 more "LPHDR execution units" and thus meets each of claims 3, 7-8, and 9-10.

5. Claims 2, 4, 6

159. Claims 2, 4, and 6 of the '273 patent recite that the "at least one first LPHDR execution unit" in the device of claims 1, 3, and 5 (respectively) "comprises at least part of an FPGA."

Claim 2
The device of claim 1, wherein the at least one first LPHDR execution unit comprises at least part of an FPGA.
Claim 4
The device of claim 3, wherein the at least one first LPHDR execution unit comprises at least part of an FPGA.
Claim 6
The device of claim 5, wherein the at least one first LPHDR execution unit comprises at least part of an FPGA.

160. Bates-2010 discloses implementing “embodiments” of its alleged “invention” on “FPGAs,” providing POSA with motivation to implement Bates-2010’s silicon-implemented transistor-based digital floating-point execution unit to “comprise[] at least part of an FPGA” in the device meeting claims 1-2, 3-4, and 5-6. Bates-2010, [0040] (“[E]mbodiments of the present invention” may use “programmable array architectures (such as *FPGAs* and *FPAAs*)”); [0163] (“[E]mbodiments of the present invention may be implemented as reconfigurable architectures, such as but not limited to programmable logic devices, field programmable analog arrays, or *field programmable gate array* architectures, such as a design in which existing multiplier blocks of an FPGA are replaced with or supplemented by LPHDR arithmetic elements of any of the kinds disclosed herein, or for example in which LPHDR elements are included in a new or existing reconfigurable device design.”), [0165] (“[E]mbodiments of the present invention may be implemented using *FPGA* or other reconfigurable chips as the underlying hardware, in which the FPGAs or other reconfigurable chips are configured to

perform the LPHDR operations disclosed herein.”). *See also* Bates-2010, [0174] (“Certain embodiments of the present invention may constitute, or may be part of, processors, which are devices capable of executing software to perform computations. ... Processors may be reconfigurable devices, such as, without limitation, *field programmable arrays*. ... Processors may be formed as a collection of component host processors and co-processors of various types, such as CPUs, GPUs, *FPGAs*, or other processors or other devices”).

6. Claims 11-17

161. Claims 11-17 of the ’273 patent depend from claim 8 and recite various minimum percentages for X and Y.

Claim 11
The device of claim 8, wherein X=10%
Claim 12
The device of claim 8, wherein Y=0.1%.
Claim 13
The device of claim 8, wherein Y=0.15%
Claim 14
The device of claim 8, wherein Y=0.2%.
Claim 15
The device of claim 8, wherein X=10% and wherein Y=0.1%.
Claim 16
The device of claim 8, wherein X=10% and wherein Y=0.15%
Claim 17
The device of claim 8, wherein X=10% and wherein Y=0.2%.

162. Bates-2010 discloses each recited minimum percentage at [0172], and explicitly suggests implementing an execution unit that achieves each claimed

combination of minimum X and Y percentages in that paragraph [0172].

Implementing the Bates-2010 device that meets claim 8 of the '273 patent with Bates-2010's silicon-implemented transistor-based digital floating-point execution units to achieve each claimed combination of minimum X and Y percentages recited in claims 11-17 of the '273 patent would have been obvious in view of Bates-2010's explicit suggestion in [0172] to achieve each of those combinations of X and Y percentages, for the same reasons I discussed in Section V.D.3.d above with respect to the X/Y combination recited in limitation [1B2].

7. Claim 18. The device of claim 8, wherein the dynamic range of the possible valid inputs to the first operation is at least as wide as from 1/1,000,000 through 1,000,000

163. *See* my discussion of limitation [1B1] above (explaining that Bates-2010's execution unit executes operations on a range of possible valid inputs from one millionth up to one million).

8. Claim 19. The device of claim 1, wherein the at least one first LPHDR execution unit comprises a plurality of locally connected LPHDR execution units

164. Bates-2010 suggests implementing a device with “only local connections between [execution] units.” Bates-2010, [0085] (“For the goal of showing usefulness, we choose a very general embodiment of an LPHDR machine. Our model of the machine is that it provides at least the following capabilities ... provides the arithmetic/memory units in a two-dimensional physical layout with

only *local connections* between units (rather than some more powerful, flexible, or sophisticated connection mechanism)....”). Implementing the Bates-2010 device I discussed in Section V.D.3 above (discussing claim 1) to have the device’s execution units locally connected, as Bates-2010 explicitly suggests, meets claim 19 of the ’273 patent.

9. Claim 20. The device of claim 1, wherein the device has a SIMD architecture

165. See my discussion of limitation [1A1] above (explaining that Bates-2010’s device is a “SIMD” processor having a SIMD architecture). Bates-2010, [0040] (“[E]mbodiments of the present invention... may be an instance of a SIMD computer architecture.”).

10. Claim 21. The device of claim 1, wherein the device includes memory locally accessible to the at least one first LPHDR execution unit

166. Bates-2010 suggests implementing execution units such that each unit has “a small amount of memory local to” it. Bates-2010, [0085] (“For the goal of showing usefulness, we choose a very general embodiment of an LPHDR machine. Our model of the machine is that it provides at least the following capabilities ... provides a small amount of *memory local to* each arithmetic unit ...”). Implementing the Bates-2010 device I discussed in Section V.D.3 above (discussing claim 1) such that the device’s execution units have memory locally

accessible to them, as Bates-2010 explicitly suggests, meets claim 21 of the '273 patent.

11. Claims 22-23

167. Claims 22-23 of the '273 patent limit the device of claim 1 by requiring that the device be “implemented on a silicon chip” (claim 22) or “implemented on a silicon chip using digital technology” (claim 23). Bates-2010 suggests implementing digital execution units using silicon fabrication technologies. Bates-2010, [0156] (referring to “a *digital* embodiment of the present invention built as a large *silicon chip*”), [0162] (“[E]mbodiments of the present invention may represent values in any of a variety of ways, such as by using *digital* or analog representations LPHDR arithmetic circuits may be implemented in any of a variety of ways, such as by using various *digital* methods (which may be parallel or serial, pipelined or not) or analog methods or combinations thereof. ... Arithmetic elements may be implemented, for example, on a single physical device, such as a *silicon chip*, or spread across multiple devices”), [0165] (“...certain embodiments of the present invention are described herein as being implemented using custom *silicon* as the hardware...embodiments of the present invention may be implemented using any programmable conventional *digital* or analog computing architecture ...”).

Implementing the Bates-2010 device discussed I discussed in Section V.D.3 above

(discussing claim 1) with Bates-2010's *silicon*-implemented transistor-based *digital* floating-point execution units, as Bates-2010 explicitly suggests, meets claims 22-23 of the '273 patent.

12. Claim 24. The device of claim 1, wherein the device further comprises a digital processor adapted to control the operation of the at least one first LPHDR execution unit

168. Bates-2010's execution units (which can include the silicon-implemented transistor-based digital floating-point execution unit I discussed above) receive "instructions" from a "central control unit," *i.e.*, "CU," that can be a "CPU" (*i.e.*, central processing unit). *See* Bates-2010, [0024], [0043], [0049]. Bates-2010 explains that "[a]rray processors distribute data across a grid of processing elements (PEs)," and that "[i]nstructions are broadcast to the PEs from a central control [unit]...." Bates-2010, [0024]. Figure 1 of Bates-2010 shows "[a]n example SIMD computing system" including "a collection of many processing elements (PEs)," which is referred to as a "Processing Element *Array* (PEA)." Bates-2010, [0043]. This system also includes a "control unit (CU)." Based on the explanation in paragraph [0024], a POSA would have understood that the CU broadcasts instructions to the PEs. Bates-2010 also explains that the "CU...may include...a CPU." Bates-2020, [0049]. A POSA would have understood that the CPU is a "digital processor" as claim 24 recites.

169. As seen in Figures 1, 4, and 6, Bates-2010's exemplary SIMD processor's "CU" sends "control signals 412a-d" to each execution unit (PE 400 and/or LPHDR arithmetic unit 408) within the device, and "[t]he operation of the PEs is controlled by [the] control signals 412a-d received from the CU 106." Bates-2010, [0047] (the CU has "the ability to issue massively parallel instructions to the [processing element array]"), [0057] ("The operation of the PEs is controlled by control signals 412a-d received from the CU 106, four of which are shown in FIG. 4 merely for purposes of example and not limitation."), [0067] ("unit 408 is controlled by control signals 412a-d").

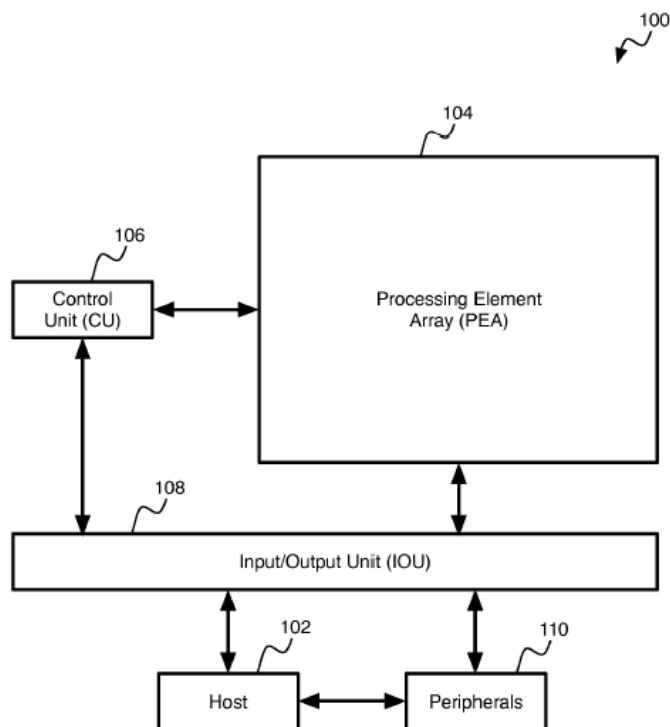


FIG. 1

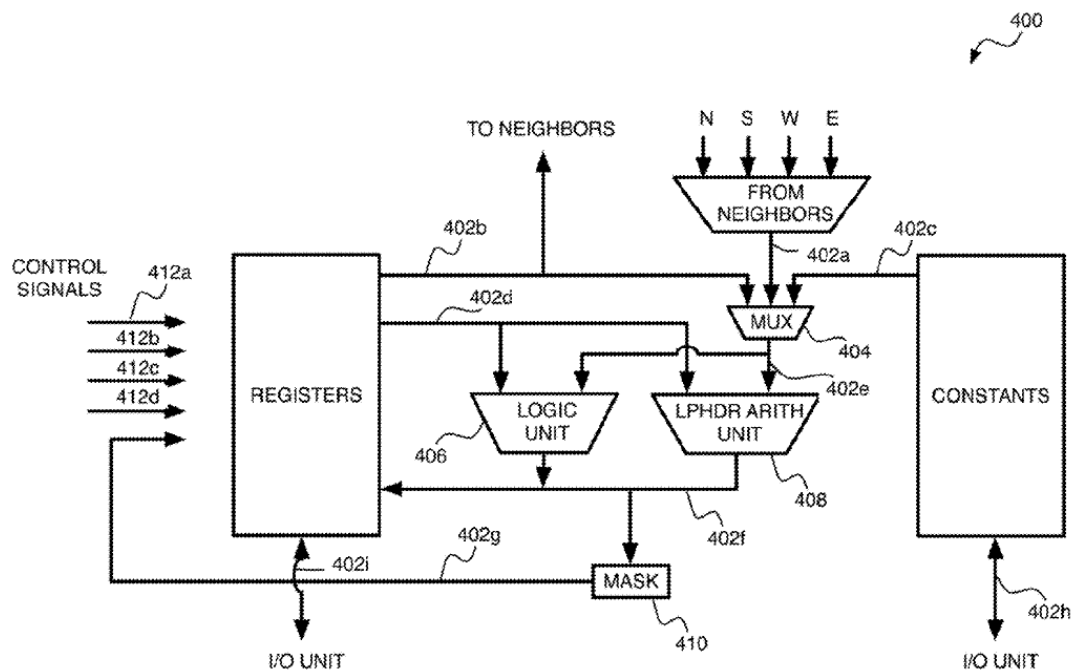


FIG. 4

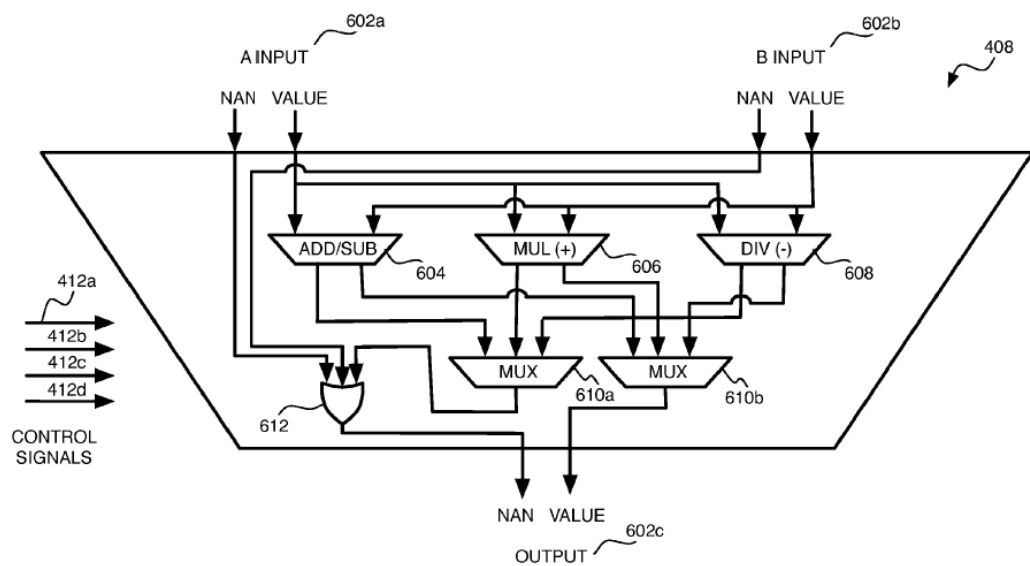


FIG. 6

170. Thus, Bates-2010 teaches including in its device a digital processor (*e.g.*, the CU implemented as a CPU) that is “adapted to control” the PEs (*i.e.*, the LPHDR execution units), meeting claim 24 of the ’273 patent.

13. Claim 25

171. Claim 25 combines limitations of claims 9, 19, and 21-23, as shown below.

Claim 25	Claim 9
25. The device of claim 1, wherein the at least one LPHDR execution unit comprises at least five hundred locally connected LPHDR execution units, wherein the device includes memory locally accessible to at least one of the LPHDR execution units, and wherein the device is implemented on a silicon chip using digital technology.	9. The device of claim 1, wherein the at least one first LPHDR execution unit comprises at least five hundred LPHDR execution units.
	Claim 19
	19. The device of claim 1, wherein the at least one first LPHDR execution unit comprises a plurality of locally connected LPHDR execution units.
	Claim 21
	21. The device of claim 1, wherein the device includes memory locally accessible to the at least one first LPHDR execution unit.
	Claim 22
	22. The device of claim 1, wherein the device is implemented on a silicon chip.
	Claim 23
	23. The device of claim 1, wherein the device is implemented on a silicon chip using digital technology.

172. As I explained in Sections V.D.4, V.D.8, and V.D.10-V.D.11 above with respect to claims 9, 19, and 21-23, Bates-2010 explicitly suggests

implementing a device including silicon-implemented transistor-based digital floating-point execution units and with the features recited in claims 9, 19, and 21-23. Combining those features as arranged in claim 25 would have been a straightforward implementation given Bates-2010's explicit suggestion to do so.

14. Claim 26. The device of claim 1, wherein the device is part of a mobile device.

173. Bates-2010 suggests using a device comprising LPHDR execution units for “mobile computing,” *i.e.*, in a “mobile device” as claim 26 of the '273 patent recites. Bates-2010, [0157] (“... a machine implemented in accordance with embodiments of the present invention can have extremely high performance with reasonable power (for instance in the tens of watts) or low power (for instance a fraction of a watt) with reasonably high performance. This means that such embodiments may be suitable for the full range of computing, from supercomputers, through desktops, down to *mobile* computing.”). Implementing the Bates-2010 device I discussed in Section V.D.3 above (discussing claim 1) as part of a mobile device, as Bates-2010 explicitly suggests, meets claim 26 of the '273 patent.

15. Claim 27. The device of claim 1, wherein the at least one first LPHDR execution unit represents numbers using a logarithmic representation

174. Bates-2010 suggests using a device comprising LPHDR execution units that represents numbers using “a logarithmic representation” as claim 27 of

the '273 patent recites. Bates-2010, [0035] (“One variety of LPHDR arithmetic represents values from one millionth up to one million with a precision of about 0.1%. If these values were represented and manipulated using the methods of floating point arithmetic ... the circuits to multiply and divide these floating point values would be relatively large. One example of an alternative embodiment is to use a *logarithmic representation* of the values.”). Implementing the Bates-2010 device I discussed in Section V.D.3 above (discussing claim 1) such that the device’s execution unit represented numbers using a “logarithmic representation,” rather than (*i.e.*, as an “alternative” to) floating-point, as Bates-2010 explicitly suggests, meets claim 27 of the '273 patent.

16. Claim 28. The device of claim 1, wherein the at least one first LPHDR execution unit represents numbers using a floating point representation

175. See my discussion of limitation [1B1] above (explaining that Bates-2010’s execution unit represents numbers using a floating-point representation).

17. Claim 29. The device of claim 1: wherein the device further comprises input means for receiving data representing an input image; and wherein the input image includes the first input signal

176. In Bates-2010, Figure 1’s exemplary SIMD device includes an “input/output unit (IOU)” that gets “data into and out of” the device. Bates-2010, [0043] (“An example SIMD computing system 100 is illustrated in FIG. 1. The computing system 100 includes... an *I/O unit (IOU) 108*”), [0048] (“In order to

get data into and out of the CU 106 and PEA 104, the *I/O Unit 108* may interface the CU 106 and PEA 104 with the Host 102, the Host's memory (not shown in FIG. 1), and the system's Peripherals 110, such as external storage (e.g., disk drives), display devices for visualization of the computational results, and sometimes special high bandwidth input devices (e.g., vision sensors). The PEA's ability to process data far faster than the Host 102 makes it useful for the IOU 108 to be able to completely bypass the Host 102 for some of its data transfers. Also, the Host 102 may have its own ways of communicating with the Peripherals 110.”), FIG. 1.

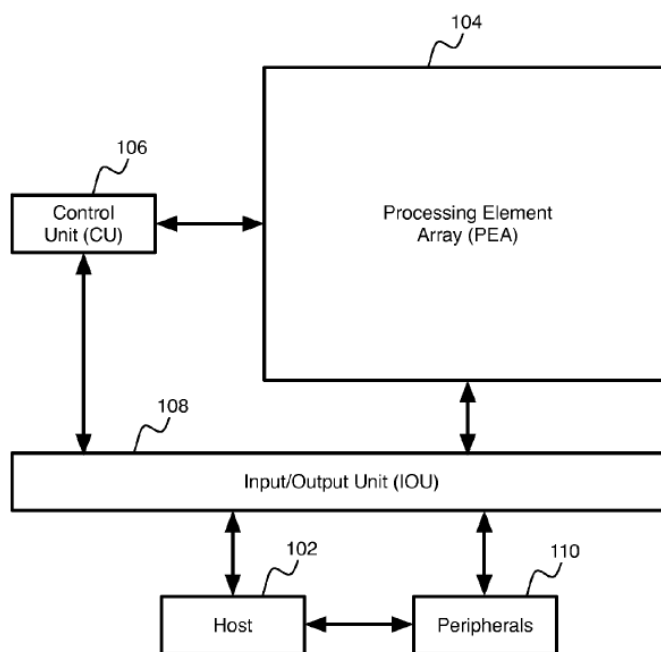


FIG. 1

Bates-2010, Figure 1

177. The “input/output unit (IOU)” is the element that the ’273 specification identifies as performing the function of “receiving data.” ’273 patent, 9:6-16 (“In order to get data into and out of the CU 106 and PEA 104, the I/O Unit 108 may interface the CU 106 and PEA 104 with the Host 102, the Host’s memory (not shown in FIG. 1), and the system’s Peripherals 110, such as external storage (e.g., disk drives), display devices for visualization of the computational results, and sometimes special high bandwidth input devices (e.g., vision sensors). The PEA’s ability to process data far faster than the Host 102 makes it useful for the IOU 108 to be able to completely bypass the Host 102 for some of its data transfers. Also, the Host 102 may have its own ways of communicating with the Peripherals 110.”); FIG. 1. Therefore, applying the standard for interpreting a “means” claim set forth in paragraph 10 of my declaration above, if an “input/output unit (IOU)” conveys a non-generic structure, then it discloses the “input means” recited in claim 29 of the ’273 patent because it is the element that the specification identifies as performing the function recited in the claim (“receiving data”).

178. Bates-2010 further discloses using a device comprising LPHDR execution units to perform deblurring of image data, which a POSA would have known involves receiving data representing an input image including the “first input signal” of the ’273 patent’s claims. Bates-2010, [0148] (“In order to gather

sufficient light to form an image, camera shutters are often left open for long enough that camera motion can cause blurring. ... If the motion path of the camera is known (or can be computed) then the blur can be substantially removed using various deblurring algorithms. One such algorithm is the Richardson-Lucy method ('RL'), and we show here that embodiments of the present invention can run that algorithm and produce useful results'), [0149] ("Algorithm. The Richardson-Lucy algorithm is well known and widely available."), [0150] ("We implemented in the C programming language a straightforward version of the RL method that uses LPHDR arithmetic."), [0151] ("We computed the mean difference, over all pixels in the image, between each original pixel value (a gray scale value from 0 to 255) and the corresponding value in the image reconstructed by the RL method."), [0152] ("the resulting deblurred images are much closer to the originals than in this case, as can be seen by shrinking the kernel length and running the RL algorithm with LPHDR arithmetic on those more typical cases"), [0153] ("It is clear to those with ordinary skill in the art that Richardson-Lucy using a local kernel performs only local computational operations. An image to be deblurred can be loaded into the PE array, storing one or more pixels per PE, the deconvolution operation of RL can then be iterated dozens or hundreds of times...."), [0154] ("An extreme form of image deblurring is the Iterative Reconstruction method used in computational

tomography. ... The method discussed above generalizes naturally to Iterative Reconstruction and makes efficient use of the machine.”).

179. A POSA would have thus understood Bates-2010 to suggest implementing the device I discussed in Section V.D.3 above (concerning claim 1) such that the device “comprises input means for receiving data representing an input image; and wherein the input image includes the first input signal” as claim 29 of the ’273 patent recites.

18. Claim 30. The device of claim 29, wherein the device is part of a mobile device

180. Implementing the Bates-2010 device discussed I discussed in Section V.D.17 above (discussing claim 29) as part of a “mobile device,” as Bates-2010 explicitly suggests, meets claim 30 of the ’273 patent. *See* Section V.D.14 above; Bates-2010, [0157] (“a machine implemented in accordance with embodiments of the present invention can have extremely high performance with reasonable power (for instance in the tens of watts) or low power (for instance a fraction of a watt) with reasonably high performance. This means that such embodiments may be suitable for the full range of computing, from supercomputers, through desktops, down to *mobile* computing.”).

19. Claim 31. The device of claim 29, wherein the device is adapted to deblur the input image

181. Implementing the Bates-2010 device discussed I discussed in Section V.D.17 above (with respect to claim 29) to be adapted to “deblur the input image,” as Bates-2010 explicitly suggests (*see* my discussion of “deblurring” in Section V.D.17 above, citing Bates-2010, [0148]-[0154]), meets claim 30 of the ’273 patent.

20. Claim 32. The device of claim 1, wherein the device is adapted to perform nearest neighbor search

182. Bates-2010 suggests using a device comprising LPHDR execution units to “find[] nearest neighbors.” Bates-2010, [0090]-[0147]. One of the “Applications” that Bates-2010 discusses for its invention is “Finding Nearest Neighbors.” Bates-2010, [0090] (under the “Findng Nearest Neighbors” heading, explaining that “[g]iven a large set of vectors, called Examples, and a given vector, called Test, the nearest neighbor problem (‘NN’) is to find the Example which is closest to Test where the distance metric is the square of the Euclidean distance (sum of squares of distances between respective components).”) Bates-2010 explains that “NN is a widely useful computation” for various tasks. Bates-2010, [0091]-[0092]. Bates-2010 describes software simulations of the nearest-neighbor problem and a related problem, distance-weighted scoring, at paragraphs [0090]-[0147]. Implementing the Bates-2010 device I discussed in Section V.D.3 above

(with respect to claim 1) to perform nearest neighbor searching, as Bates-2010 explicitly suggests, meets claim 32 of the '273 patent.

21. Claims 36-67

183. Limitations [36A1]-[36B2] of independent claim 36 of the '273 patent are identical to limitations [1A1]-[1B2] of claim 1, and are met for the reasons I discussed in Section V.D.3 above with respect to claim 1. Limitation [36C] does not appear in claim 1, but is identical to claim 5, except that limitation [36C] is broader than claim 5 because [36C] omits the “by at least ten” that claim 5 recites. Thus, limitation [36C] is met for the same reasons I discussed in Section V.D.4 above with respect to claim 5.

Claim 36	Claims 1 and 5
[36A1] A device: comprising at least one first low precision high-dynamic range (LPHDR) execution unit	[1A1] A device: comprising at least one first low precision high dynamic range (LPHDR) execution unit
[36A2] adapted to execute a first operation on a first input signal representing a first numerical value to produce a first output signal representing a second numerical value,	[1A2] adapted to execute a first operation on a first input signal representing a first numerical value to produce a first output signal representing a second numerical value,
[36B1] wherein the dynamic range of the possible valid inputs to the first operation is at least as wide as from 1/65,000 through 65,000 and	[1B1] wherein the dynamic range of the possible valid inputs to the first operation is at least as wide as from 1/65,000 through 65,000 and
[36B2] for at least X=5% of the possible valid inputs to the first operation, the statistical mean, over repeated execution of the first operation on each specific input from the at least X % of the possible valid inputs to the first operation, of the	[1B2] for at least X=5% of the possible valid inputs to the first operation, the statistical mean, over repeated execution of the first operation on each specific input from the at least X % of the possible valid inputs to the first operation, of the numerical values

numerical values represented by the first output signal of the LPHDR unit executing the first operation on that input differs by at least $Y=0.05\%$ from the result of an exact mathematical calculation of the first operation on the numerical values of that same input;	represented by the first output signal of the LPHDR unit executing the first operation on that input differs by at least $Y=0.05\%$ from the result of an exact mathematical calculation of the first operation on the numerical values of that same input.
[36C] wherein the number of LPHDR execution units in the device <i>exceeds the non-negative integer number</i> of execution units in the device adapted to execute at least the operation of multiplication on floating point numbers that are at least 32 bits wide.	5. The device of claim 1, wherein the number of LPHDR execution units in the device <i>exceeds by at least ten the non-negative integer number</i> of execution units in the device adapted to execute at least the operation of multiplication on floating point numbers that are at least 32 bits wide.

184. Claims 37-67 depend from claim 36, but otherwise are identical to claims 2-32, respectively, which depend from claim 1. The limitations recited in claims 37-67 are thus met for the reasons I discussed in Sections V.D.4-20 above with respect to claims 2-32.

Claim 2	Claim 37
The device of claim 1, wherein the at least one first LPHDR execution unit comprises at least part of an FPGA	The device of claim 36, wherein the at least one first LPHDR execution unit comprises at least part of an FPGA.
Claim 3	Claim 38
The device of claim 1, wherein the at least one first LPHDR execution unit comprises at least ten LPHDR execution units.	The device of claim 36, wherein the at least one first LPHDR execution unit comprises at least ten LPHDR execution units.
Claim 4	Claim 39
The device of claim 3, wherein the at least one first LPHDR execution unit comprises at least part of an FPGA.	The device of claim 38, wherein the at least one first LPHDR execution unit comprises at least part of an FPGA.

Claim 5	Claim 40
The device of claim 1, wherein the number of LPHDR execution units in the device exceeds by at least ten the non-negative integer number of execution units in the device adapted to execute at least the operation of multiplication on floating point numbers that are at least 32 bits wide.	The device of claim 36, wherein the number of LPHDR execution units in the device exceeds by at least ten the non-negative integer number of execution units in the device adapted to execute at least the operation of multiplication on floating point numbers that are at least 32 bits wide.
Claim 6	Claim 41
The device of claim 5, wherein the at least one first LPHDR execution unit comprises at least part of an FPGA.	The device of claim 40, wherein the at least one first LPHDR execution unit comprises at least part of an FPGA.
Claim 7	Claim 42
The device of claim 1, wherein the at least one first LPHDR execution unit comprises at least one hundred LPHDR execution units.	The device of claim 36, wherein the at least one first LPHDR execution unit comprises at least one hundred LPHDR execution units.
Claim 8	Claim 43
The device of claim 1, wherein the number of LPHDR execution units in the device exceeds by at least one hundred the non-negative integer number of execution units in the device adapted to execute at least the operation of multiplication on floating point numbers that are at least 32 bits wide.	The device of claim 36, wherein the number of LPHDR execution units in the device exceeds by at least one hundred the non-negative integer number of execution units in the device adapted to execute at least the operation of multiplication on floating point numbers that are at least 32 bits wide.
Claim 9	Claim 44
The device of claim 1, wherein the at least one first LPHDR execution unit comprises at least five hundred LPHDR execution units.	The device of claim 36, wherein the at least one first LPHDR execution unit comprises at least five hundred LPHDR execution units.
Claim 10	Claim 45
The device of claim 1, wherein the number of LPHDR execution units in the device exceeds by at least five hundred the non-negative integer number of execution units in the device	The device of claim 36, wherein the number of LPHDR execution units in the device exceeds by at least five hundred the non-negative integer number of execution units in the device

adapted to execute at least the operation of multiplication on floating point numbers that are at least 32 bits wide.	adapted to execute at least the operation of multiplication on floating point numbers that are at least 32 bits wide.
Claim 11	Claim 46
The device of claim 8, wherein X=10%.	The device of claim 43, wherein X=10%.
Claim 12	Claim 47
The device of claim 8, wherein Y=0.1%.	The device of claim 43, wherein Y=0.1%.
Claim 13	Claim 48
The device of claim 8, wherein Y=0.15%.	The device of claim 43, wherein Y=0.15%.
Claim 14	Claim 49
The device of claim 8, wherein Y=0.2%.	The device of claim 43, wherein Y=0.2%.
Claim 15	Claim 50
The device of claim 8, wherein X=10% and wherein Y=0.1%.	The device of claim 43, wherein X=10% and wherein Y=0.1%.
Claim 16	Claim 51
The device of claim 8, wherein X=10% and wherein Y=0.15%.	The device of claim 43, wherein X=10% and wherein Y=0.15%.
Claim 17	Claim 52
The device of claim 8, wherein X=10% and wherein Y=0.2%.	The device of claim 43, wherein X=10% and wherein Y=0.2%.
Claim 18	Claim 53
The device of claim 8, wherein the dynamic range of the possible valid inputs to the first operation is at least as wide as from 1/1,000,000 through 1,000,000.	The device of claim 43, wherein the dynamic range of the possible valid inputs to the first operation is at least as wide as from 1/1,000,000 through 1,000,000.
Claim 19	Claim 54
The device of claim 1, wherein the at least one first LPHDR execution unit comprises a plurality of locally connected LPHDR execution units.	The device of claim 36, wherein the at least one first LPHDR execution unit comprises a plurality of locally connected LPHDR execution units.
Claim 20	Claim 55
The device of claim 1, wherein the device has a SIMD architecture.	The device of claim 36, wherein the device has a SIMD architecture.

Claim 21	Claim 56
The device of claim 1, wherein the device includes memory locally accessible to the at least one first LPHDR execution unit.	The device of claim 36, wherein the device includes memory locally accessible to the at least one first LPHDR execution unit.
Claim 22	Claim 57
The device of claim 1, wherein the device is implemented on a silicon chip.	The device of claim 36, wherein the device is implemented on a silicon chip.
Claim 23	Claim 58
The device of claim 1, wherein the device is implemented on a silicon chip using digital technology.	The device of claim 36, wherein the device is implemented on a silicon chip using digital technology.
Claim 24	Claim 59
The device of claim 1, wherein the device further comprises a digital processor adapted to control the operation of the at least one first LPHDR execution unit.	The device of claim 36, wherein the device further comprises a digital processor adapted to control the operation of the at least one first LPHDR execution unit.
Claim 25	Claim 60
The device of claim 1, wherein the at least one LPHDR execution unit comprises at least five hundred locally connected LPHDR execution units, wherein the device includes memory locally accessible to at least one of the LPHDR execution units, and wherein the device is implemented on a silicon chip using digital technology.	The device of claim 1, wherein the at least one LPHDR execution unit comprises at least five hundred locally connected LPHDR execution units, wherein the device includes memory locally accessible to at least one of the LPHDR execution units, and wherein the device is implemented on a silicon chip using digital technology.
Claim 26	Claim 61
The device of claim 1, wherein the device is part of a mobile device.	The device of claim 36, wherein the device is part of a mobile device.
Claim 27	Claim 62
The device of claim 1, wherein the at least one first LPHDR execution unit represents numbers using a logarithmic representation	The device of claim 36, wherein the at least one first LPHDR execution unit represents numbers using a logarithmic representation

Claim 28	Claim 63
The device of claim 1, wherein the at least one first LPHDR execution unit represents numbers using a floating point representation.	The device of claim 36, wherein the at least one first LPHDR execution unit represents numbers using a floating point representation.
Claim 29	Claim 64
The device of claim 1: wherein the device further comprises input means for receiving data representing an input image; and wherein the input image includes the first input signal.	The device of claim 36: wherein the device further comprises input means for receiving data representing an input image; and wherein the input image includes the first input signal.
Claim 30	Claim 65
The device of claim 29, wherein the device is part of a mobile device	The device of claim 64, wherein the device is part of a mobile device
Claim 31	Claim 66
The device of claim 29, wherein the device is adapted to deblur the input image.	The device of claim 64, wherein the device is adapted to deblur the input image.
Claim 32	Claim 67
The device of claim 1, wherein the device is adapted to perform nearest neighbor search	The device of claim 36, wherein the device is adapted to perform nearest neighbor search

22. Claims 33 and 68

185. Independent claim 33 of the '273 patent recites a “device comprising a computer processor and a computer-readable memory storing computer program instructions, wherein the computer program instructions are executable by the processor to emulate a second device,” and describes the second device using language identical to claim 1 and its claimed “device.”

Claim 33	Claim 1
[33A1] A device comprising a computer processor and a computer-readable memory storing computer program instructions, wherein the computer program instructions are executable by the processor to emulate <i>a second device comprising: at least one first low precision high-dynamic range (LPHDR) execution unit</i>	[1A1] <i>A device: comprising at least one first low precision high dynamic range (LPHDR) execution unit</i>
[33A2] adapted to execute a first operation on a first input signal representing a first numerical value to produce a first output signal representing a second numerical value;	[1A2] adapted to execute a first operation on a first input signal representing a first numerical value to produce a first output signal representing a second numerical value;
[33B1] wherein the dynamic range of the possible valid inputs to the first operation is at least as wide as from 1/65,000 through 65,000 and	[1B1] wherein the dynamic range of the possible valid inputs to the first operation is at least as wide as from 1/65,000 through 65,000 and
[33B2] for at least X=5% of the possible valid inputs to the first operation, the statistical mean, over repeated execution of the first operation on each specific input from the at least X % of the possible valid inputs to the first operation, of the numerical values represented by the first output signal of the LPHDR unit executing the first operation on that input differs by at least Y=0.05% from the result of an exact mathematical calculation of the first operation on the numerical values of that same input.	[1B2] for at least X=5% of the possible valid inputs to the first operation, the statistical mean, over repeated execution of the first operation on each specific input from the at least X % of the possible valid inputs to the first operation, of the numerical values represented by the first output signal of the LPHDR unit executing the first operation on that input differs by at least Y=0.05% from the result of an exact mathematical calculation of the first operation on the numerical values of that same input.

186. Independent claim 68 similarly recites a “device comprising a computer processor and a computer-readable memory storing computer program instructions, wherein the computer program instructions are executable by the

processor to emulate a second device,” and describes the second device using language identical to claim 36 and its claimed “device.”

Claim 68	Claim 36
[68A1] A device comprising a computer processor and a computer-readable memory storing computer program instructions, wherein the computer program instructions are executable by the processor to emulate <i>a second device comprising: at least one first low precision high-dynamic range (LPHDR) execution unit</i>	[36A1] <i>A device: comprising at least one first low precision high-dynamic range (LPHDR) execution unit</i>
[68A2] adapted to execute a first operation on a first input signal representing a first numerical value to produce a first output signal representing a second numerical value,	[36A2] adapted to execute a first operation on a first input signal representing a first numerical value to produce a first output signal representing a second numerical value,
[68B1] wherein the dynamic range of the possible valid inputs to the first operation is at least as wide as from 1/65,000 through 65,000 and	[36B1] wherein the dynamic range of the possible valid inputs to the first operation is at least as wide as from 1/65,000 through 65,000 and
[68B2] for at least 5% of the possible valid inputs to the first operation, the statistical mean, over repeated execution of the first operation on each specific input from the at least 5% of the possible valid inputs to the first operation, of the numerical values represented by the first output signal of the LPHDR unit executing the first operation on that input differs by at least 0.05% from the result of an exact mathematical calculation of the first operation on the numerical values of that same input;	[36B2] for at least X=5% of the possible valid inputs to the first operation, the statistical mean, over repeated execution of the first operation on each specific input from the at least X % of the possible valid inputs to the first operation, of the numerical values represented by the first output signal of the LPHDR unit executing the first operation on that input differs by at least Y=0.05% from the result of an exact mathematical calculation of the first operation on the numerical values of that same input;
[68C] wherein the number of LPHDR execution units in the second device exceeds the non-negative integer	[36C] wherein the number of LPHDR execution units in the device exceeds the non-negative integer number of

number of execution units in the second device adapted to execute at least the operation of multiplication on floating point numbers that are at least 32 bits wide.	execution units in the device adapted to execute at least the operation of multiplication on floating point numbers that are at least 32 bits wide.
--	---

187. Bates-2010 discloses that “embodiments of the present invention may be implemented using” a “conventional” computer “which has been programmed with software,” *e.g.*, “a software emulator,” “to perform the LPHDR operations disclosed herein.” Bates-2010, [0165] (“[E]mbodiments of the present invention may be implemented using any programmable conventional digital or analog computing architecture...which has been programmed with software to perform the LPHDR operations disclosed herein. For example, embodiments of the present invention may be implemented using a software emulator of the functions disclosed herein.”). A POSA would have understood that emulation software was typically run on computers, and that the standard configuration for a computer included a computer-readable memory storing computer programs, including the emulation program, and a processor to execute that executes instructions in computer programs.

188. Thus, given this teaching and a POSA’s knowledge, a POSA would have had reason to utilize a computer device comprising a processor and computer-readable memory storing computer program instructions (software) to emulate Figure 1’s SIMD processor implemented using Bates-2010’s silicon-implemented

transistor-based digital floating-point LPHDR execution unit(s) that meet claims 1 and 36 (as I discussed in Sections V.D.3 and V.D.21 above), thus meeting claims 33 and 68 of the '273 patent.

23. Claims 34-35, 69-70

189. As shown below, claims 34 and 69 of the '273 patent are identical to claim 3, except that claims 34 and 69 depend from claims 33 and 68, respectively, and further limit the emulated “second” device, whereas claim 3 depends from claim 1 and limits claim 1’s device. Claims 35 and 70 are identical to claim 5, except that claims 35 and 70 depend from claims 33 and 68, respectively, and further limit the emulated “second” device, whereas claim 5 depends from claim 1 and limits claim 1’s device.

Claim 3	Claim 34	Claim 69
The device of claim 1, wherein the at least one first LPHDR execution unit comprises at least ten LPHDR execution units.	The device of claim 33, wherein the at least one first LPHDR execution unit comprises at least ten LPHDR execution units.	The device of claim 68, wherein the at least one first LPHDR execution unit comprises at least ten LPHDR execution units.
Claim 5	Claim 35	Claim 70
The device of claim 1, wherein the number of LPHDR execution units in the device exceeds by at least ten the non-negative integer number of execution units in the device adapted to execute at least the operation of multiplication on floating	The device of claim 33, wherein the number of LPHDR execution units in the second device exceeds by at least ten the non-negative integer number of execution units in the second device adapted to execute at least the operation of	The device of claim 68, wherein the number of LPHDR execution units in the second device exceeds by at least ten the non-negative integer number of execution units in the second device adapted to execute at least the operation of

point numbers that are at least 32 bits wide.	multiplication on floating point numbers that are at least 32 bits wide.	multiplication on floating point numbers that are at least 32 bits wide.
---	--	--

190. Bates-2010 meets claims 34-35 and 69-70 of the '273 patent because Bates-2010 suggests emulating the device it discloses, and Bates-2010 discloses the device recited in each of claims 34-35 and 60-70 for the same reasons discussed above in connection with the identical limitations in claims 3 and 5. *See* Bates-2010, [0165] (“embodiments of the present invention may be implemented using any programmable conventional digital or analog computing architecture (including those which use high-precision computing elements, including those which use other kinds of non-LPHDR hardware to perform LPHDR arithmetic, and including those which are massively parallel) which has been programmed with software to perform the LPHDR operations disclosed herein. For example, embodiments of the present invention may be implemented using a software emulator of the functions disclosed herein.”); *see* Section V.D.4 above.

VI. CLAIMS 1-2, 21-24, 26, AND 28 WOULD HAVE BEEN OBVIOUS OVER DOCKSER

191. The '273 patent admits that LPHDR execution units were not novel—it alleges that they were known but were considered to be “not useful” for performing “massive amounts” of computations. '273 patent, 6:58-7:11 (“it is commonly believed by those having ordinary skill in the art, that LPHDR computation, and in particular massive amounts of LPHDR computation ... is not practical as a substrate for moderately general computing ... Despite these views—that massive amounts of arithmetic on a chip or in a massively parallel machine are not useful, and that massive amounts of LPHDR arithmetic are even worse—embodiments of the present invention disclosed herein demonstrate that massively parallel LPHDR designs are in fact useful ...”).

192. The '273 patent purports to encompass as its alleged invention the idea “that LPHDR arithmetic is useful in several important practical computing applications” to save power. '273 patent, 16:25-27, 23:65-24:11. However, the '273 patent's claims are not limited to methods of using LPHDR execution units for any *particular* computing applications. Instead, as I discussed above, all the independent claims recite a device that includes an LPHDR execution unit. But a device including an LPHDR execution unit was not new. For example, as I demonstrate below, Dockser (Ex. 1007) disclosed a device including an LPHDR execution unit as the '273 patent claims.

A. Dockser (Ex. 1007)

193. Dockser discloses performing “mathematical operations” including “multiplication” using a “floating-point processor having a given precision.”

Dockser, Abstract (“A method and apparatus for performing a floating-point operation with a floating-point processor having a given precision is disclosed. A subprecision for the floating-point operation on one or more floating-point numbers is selected.”), [0001] (“Floating-point processors are specialized computing units that perform certain mathematical operations, e.g., multiplication, division, trigonometric functions, and exponential functions, at high speed.”).

194. “Floating-point” is a known number representation that uses “a sign component, an exponent, and a mantissa.” Dockser, [0001] (“A floating-point representation of a number commonly includes a sign component, an exponent, and a mantissa.”). The sign indicates whether the number is positive or negative, and the number’s absolute value is equal to the mantissa multiplied by a base raised to the power of the exponent. Dockser, [0001] (“To find the value of a floating-point number, the mantissa is multiplied by a base (commonly 2 in computers) raised to the power of the exponent. The sign is applied to the resultant value.”). Floating-point representation was widely used in computing as of the time of the ’273 patent’s alleged invention. For example, the ’273 patent refers in

its “Background” section to the “widely used IEEE 754 single precision floating point standard.” ’273 patent, 2:32-33.

195. In the computing arts, the term “floating-point” is conventionally understood to describe a base-two (i.e., binary) representation unless some other base is expressly specified (e.g., decimal floating-point). That is how I use the term “floating point” in this declaration, and how the term is used in the ’273 patent and in all the prior-art references cited in this declaration. *See, e.g.,* ’273 patent, 16:6-8 (“To get sufficient dynamic range in such an embodiment, the *floating point numbers* may be processed as base 4 numbers, rather than *the usual base 2 numbers.*”); Dockser, [0001] (“To find the value of a floating-point number, the mantissa is multiplied by *a base (commonly 2 in computers)* raised to the power of the exponent.”), Tong (Ex. 1008), 274 (“There are two different IEEE standards for FP computation. IEEE 754 is a binary standard that requires the implied radix (base) to be two. IEEE 854 allows either radix 2 or radix 10 representation. By far, IEEE 754 is more popular and most desktop microprocessors support the IEEE 754 standard.”). Binary floating-point is a base-2 cousin of the familiar base-10 scientific notation, which represents, *e.g.*, the number 3,046 as 3.046×10^3 . For evidence corroborating that this was background knowledge for a POSA, *see, e.g.*, Dockser-Lall (Ex. 1010), [0003] (“Multiplying the significand by 2 raised to an integer exponent is the binary

analog to scientific notation in the base 10 system. That is, the value of the exponent determines the number of bit positions, and the direction, that the binary point in the significand should be shifted to realize the actual numerical value—hence the term, floating point.”).

196. An example of a floating-point representation represents the number 6 as: positive sign, exponent=2, mantissa=1.5 (*i.e.*, $1.5 \times 2^2 = 6$). As another example, the floating-point representation of the number –10 is negative sign, exponent=3, mantissa=1.25 (*i.e.*, $-1.25 \times 2^3 = -10$).

197. In standard floating-point representations of “normal” numbers (which are anything not smaller than 2^{-126}), the mantissa is kept in the range $1 \leq \text{mantissa} < 2$, so only the fraction by which the mantissa exceeds 1 needs to be specified (*i.e.*, the leading 1 in the mantissa is “implied”). *See, e.g.*, Dockser, [0002], explaining that in the IEEE-754 “32-bit single format having a 1-bit sign, an 8-bit exponent, and a 23-bit mantissa,” “[o]nly the 23 fraction bits of the mantissa are stored in the 32-bit encoding, an integer bit, immediately to the left of the binary point, is implied.” Further evidence corroborating the POSA’s background knowledge in this regard can be found, for example, in Dockser-Lall (Ex. 1010), which shares one of Dockser’s inventors. Dockser-Lall explains that “[w]hen the significand is in the range $1 \leq \text{significand} < 2$ and the exponent is within its defined range, the floating-point value is referred to as a ‘normal’

number.” Dockser-Lall, [0004]. The word “significand” is another word for “mantissa.” Dockser-Lall, [0003] (“A floating-point number comprises a fixed-point *significand (also known as a mantissa)* multiplied by the base 2 raised to an integer exponent.”). Dockser-Lall explains that “[t]he significand of a normal floating-point number is thus of the form *1.fraction*, where ‘fraction’ is a binary value representing the fractional portion of the significand greater than one.” Dockser-Lall, [0004]. Since the 23 stored mantissa bits in the single-format number represent the fraction, as I explained above, this means that the “hidden”/“implied” bit has a value of 1, to make the mantissa be of the form “*1.fraction*.” Dockser-Lall, [0004]. A POSA would have understood that the smallest possible value of a “normal” number is 2^{-126} because the smallest possible normal mantissa value is 1, and the smallest exponent value is -126. Dockser-Lall, [0004] (“When the significand is in the range $1 \leq \textit{significand} < 2$ and the exponent is within its defined range, the floating-point value is referred to as a ‘normal’ number...In the IEEE 754 standard, the value of the exponent for a single-precision floating-point number ranges from -126 to 127 .”).

198. As an example of how “normal” mantissas are represented, if three bits are used in the representation, the mantissa 1.5 can be represented with three bits (100) that represent the fraction .5 as $(1 \times 2^{-1}) + (0 \times 2^{-2}) + (0 \times 2^{-3}) = .5$, and the mantissa 1.25 can be represented as the bit sequence 010 that represents

the fraction .25 as $(0 \times 2^{-1}) + (1 \times 2^{-2}) + (0 \times 2^{-3}) = .25$. POSAs understood that each bit of the mantissa fraction represents a successively smaller power of 2, starting with in the left-most fraction bit.

199. Dockser’s “Background” explains the well-known principle of mathematics and computing that “precision . . . is defined by the number of bits used to represent the mantissa”—“[t]he more bits in the mantissa, the greater the precision.” Dockser, [0002] (“The precision of the floating-point processor is defined by the number of bits used to represent the mantissa. The more bits in the mantissa, the greater the precision.”). For additional evidence corroborating that this was background knowledge for a POSA, *see, e.g.*, Gaffar (Ex. 1012), 4 (“In the case of floating-point, the precision depends on the mantissa bitwidth, while the range depends on the exponent bit-width.”); Hekstra (Ex. 1013), 4:47-52 (“A definition of floating-point numbers is given. A floating-point number x is represented by a tuple (e_x, m_x) with...mantissa[:]. An N_m -bit, signed, mantissa m_x . The number of bits N_m , not including the sign bit, determine the precision of the representation.”).

200. For example, the floating-point representation of the number 29 is positive sign, exponent=4, mantissa=1.8125 (*i.e.*, $1.8125 \times 2^4 = 29$). The 1.8125 mantissa can be represented precisely as the 4-bit sequence 1101, which represents the fraction .8125 as $(1 \times 2^{-1}) + (1 \times 2^{-2}) + (0 \times 2^{-3}) + (1 \times 2^{-4}) = .8125$.

However, if only three bits are used, then the closest mantissas that can be represented are 1.75 or 1.875. That is, the bit sequence 110 represents a mantissa of $1 + (1 \times 2^{-1}) + (1 \times 2^{-2}) + (0 \times 2^{-3}) = 1.75$, while the bit sequence 111 represents a mantissa of $1 + (1 \times 2^{-1}) + (1 \times 2^{-2}) + (1 \times 2^{-3}) = 1.875$. Thus, with a 3-bit mantissa, the number 29 cannot be represented precisely, but can only be approximated less precisely as $1.75 \times 2^4 = 28$ or as $1.875 \times 2^4 = 30$.

201. Dockser explains, as was well known, that “modern computers” “commonly” use the “ANSI/IEEE-754 standard... 32-bit single format,” which represents floating-point numbers using “a 1-bit sign, an 8-bit exponent, and a 23-bit mantissa.” Dockser, [0002]. Dockser recognized, however, that “[w]hile some applications may require these types of precision, other applications may not.” Dockser, [0003]. For example, if the lower precision from using just 16 bits in the mantissa (vs. the IEEE-754 standard 23-bit mantissa) results in computations whose accuracy suffices for a particular application (*e.g.*, generating 3D graphics), then “any accuracy beyond 16 bits of precision tends to result in unnecessary power consumption.” Dockser, [0003]. Dockser teaches that “[t]his is of particular concern in battery operated devices where power comes at a premium.” Dockser, [0003].

202. Dockser’s “floating-point processor (FPP)” operates at a selectable “subprecision of the floating-point format,” below the precision of the “IEEE-754

32-bit single format” of the input numbers it receives, by removing power from some of the least-significant mantissa bits in the FPP’s operation, resulting in those mantissa bits being dropped and “reduc[ing] the [FPP’s] power consumption.”

Dockser, [0014]-[0018], [0026]-[0027]. Dockser states that “In at least one embodiment of a floating-point processor, the precision for one or more floating-point operations may be reduced from that of the specified format, and that “[b]y selecting the subprecision of the floating-point format, to that needed for a particular operation, thereby reducing the power consumption of the floating-point processor to support the selected subprecision, greater efficiency as well as significant power savings can be achieved.” Dockser, [0014]. Dockser describes a “floating-point processor (FPP) 100 with selectable subprecision.” Dockser, [0015]. The FPP “includes a floating-point register file (FPR) 110; a floating-point controller (CTL) 130; and a floating-point mathematical operator (FPO) 140.” Dockser, [0015]. The register file includes “addressable register locations ... each configured to store an operand for a floating-point operation.” Dockser, [0016]. “Each register location 200 is configured to store a 32-bit binary floating-point number, in an IEEE-754 32-bit single format.” Dockser, [0017]. A “control register (CRG) 137” in the “floating-point controller 130” “may be loaded with subprecision select bits,” which “may be used by the floating-point controller 130 to reduce the precision of the operands.” Dockser, [0018]. The operand precision

reduction is accomplished by “caus[ing] power to be removed from the floating-point register elements for the excess bits of the fraction that are not required to meet the precision specified by the subprecision select bits,” (Dockser, [0026]), as I explain in Section VI.B.4.c(1) below. “The subprecision select bits may also be used to remove power from logic in the floating-point operator FPO 140 that is not used when the selected subprecision is reduced,” (Dockser, [0018]), as I explain in Section VI.B.4.c(2) below. *See also* Dockser, [0027] (“power savings may be achieved by removing power to the logic in the floating-point operator 140 that remains unused as a result of the subprecision selected”).

203. As I explain in detail below, certain implementation details (*e.g.*, the range of numbers conventionally represented using the 32-bit IEEE-754 standard) were so well known as to be unnecessary for Dockser to explain. *See, e.g.*, ’273 patent, 2:32-33 (referring to “widely used IEEE 754 single precision floating point standard”). In the sections below I explain that a POSA would have understood Dockser to teach a device implemented in these ways, or alternatively that they would have been the obvious implementations of Dockser’s teachings. When I refer below to Dockser “meeting” claim limitations, I am referring to Dockser’s device implemented to include any details discussed below as being the understood and/or obvious implementation of what Dockser describes.

B. Claim 1

1. [1A1] A device comprising: at least one first low precision high dynamic range (LPHDR) execution unit

204. Dockser's FPP is "incorporate[d]" in a "computing system[...], either as part of the main processor or as a coprocessor. Dockser, [0001], [0015], [0035]-[0036]. Dockser explains that "computing systems often incorporate floating-point processors, either as part of the main processor or as a coprocessor." Dockser, [0001]. Dockser's FPP likewise is a "floating-point processor (FPP)" that "may be implemented as part of the main processor, a coprocessor, or a separate entity connected to the main processor through a bus or other channel." Dockser, [0015]. And Dockser states that "[t]he various illustrative logical units, blocks, modules, circuits, elements, and/or components described in connection with the embodiments disclosed herein may be implemented or performed in a floating-point processor that is part of a general purpose processor." Dockser, [0035]. Furthermore, Dockser states that its "methods or algorithms... may be embodied directly in hardware, in a software module executed by a processor, or in a combination of the two." Dockser, [0036]. Thus, a POSA would have understood Dockser to disclose that its FPP can be part of a computing system as a main processor or as a coprocessor.

205. A POSA would have understood that the "computing system" that the FPP is incorporated into is a claimed "*device*" *comprising* the FPP. The '273

patent uses the term “device” to include “a computer including a processor and other components” or “[m]ore generally, any device... which performs the functions disclosed.” *See, e.g.*, ’273 patent, 29:5-16 (“Embodiments of the present invention may, however, be implemented in devices in addition to or other than processors ...More generally, any device or combination of devices, whether or not falling within the meaning of a “processor,” which performs the functions disclosed herein may constitute an example of an embodiment of the present invention.”), 1:61 (“Real computers are built as physical devices...”), 24:10-11 (referring to “conventional computing devices”), 27:63-66 (“For certain *devices (such as computers or processors or other devices) embodied according the present invention*, the number of LPHDR arithmetic elements in the device (e.g., *computer or processor or other device*) exceeds...”), 7:40-43 (“Various computing *devices* implemented according to embodiments of the present invention will now be described. Some of these embodiments may be an instance of a *SIMD computer* architecture.”).

206. Furthermore, within Dockser’s computing system, the FPP is implemented in a “hardware component[]” such as an “ASIC” or “FPGA,” which is also a “device” (as recited in limitation [1A1]) comprising the FPP. Dockser, [0035] (“The various illustrative logical units, blocks, modules, circuits, elements, and/or components described *in connection with the embodiments disclosed*

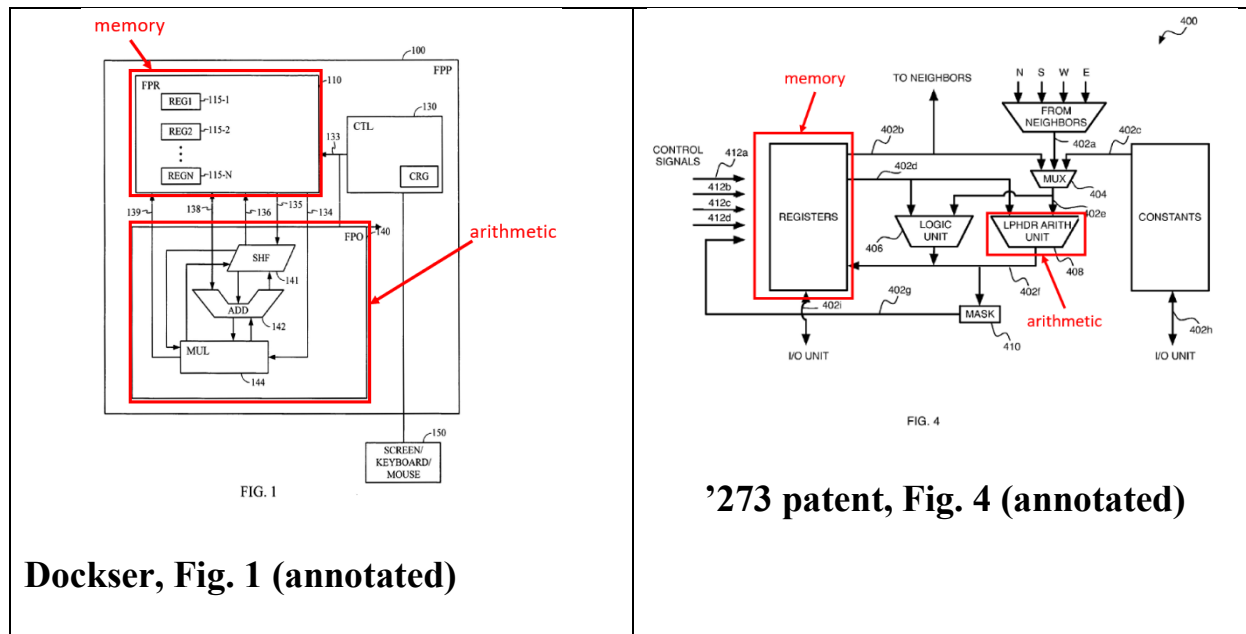
herein may be implemented or performed in a floating-point processor that is part of a general purpose processor, a digital signal processor (DSP), an application specific integrated circuit (ASIC), a field programmable gate array (FPGA) or other programmable logic component, discrete gate or transistor logic, discrete hardware components, or any combination thereof designed to perform the functions described herein.”); ’273 patent, 24:65-25:22 (“Arithmetic elements may be implemented, for example, *on a single physical device*, such as a silicon chip, or spread across multiple devices *For example*, embodiments of the present invention may be implemented as reconfigurable architectures, such as but not limited to programmable logic devices, field programmable analog arrays, or field programmable gate array architectures, such as a design in which existing multiplier blocks of an FPGA are replaced with or supplemented by LPHDR arithmetic elements of any of the kinds disclosed herein, or for example in which LPHDR elements are included in *a new or existing reconfigurable device* design.”).

207. A POSA would have understood that Dockser’s FPP is an “*execution unit*” as claimed. It is a “specialized computing unit[] that perform[s] certain mathematical operations, e.g., multiplication, [etc.]... at high speed” by “execut[ing]... instructions” including “add and subtract instructions” and “multiply instructions.” Dockser, [0001] (“Floating-point processors are

specialized computing units that perform certain mathematical operations, e.g., multiplication...at high speed.”), [0019] (“The floating-point operator 140” in Dockser’s FPP “may include one or more components configured to perform the floating-point operations. These components may include, but are not limited to, computational units such as... a floating-point multiplier (MUL) 144 configured to execute floating-point multiply instructions.”), [0024] (“Upon receiving the operands from the floating-point register file 110, one or more computational units in the floating-point operator 140 may execute the instructions of the requested floating-point operation on the received operands, at the subprecision selected by the floating-point controller 130.”).

208. The ’273 patent describes a “processing element” (PE) as one possible embodiment of an LPHDR “execution unit.” ’273 patent, 8:7-11 (“references herein to ‘*processing elements*’ within embodiments of the present invention should be *understood more generally as any kind of execution unit*, whether for performing LPHDR operations or otherwise”), 8:25-28 (“One embodiment of the present invention is a SIMD computing system of the kind shown in FIG. 1, in which one or more (e.g., all) of *the PEs* [processing elements] in the PEA 104 *are LPHDR elements*, as that term is used herein.”). The ’273 patent’s PE is a “unit[] which pairs memory with arithmetic” and implements the memory as registers. ’273 patent, 16:54-56 (“for the purpose of discussion below, we call each unit,

which pairs memory with arithmetic, a Processing Element or ‘PE’); FIG. 4 (reproduced below and annotated). Dockser’s FPP is also a unit which pairs memory with arithmetic and implements the memory as registers. Dockser, FIG. 1 (reproduced below and annotated).



209. As the annotated figures above illustrate, both Dockser’s FPP and the ’273 patent’s PE include registers that locally store data and an arithmetic unit that performs arithmetic operations on data stored in the registers, as I explain in paragraphs 210-215 below.

210. Figure 4 of the ’273 patent “shows an example design for a PE [processing element] 400.” ’273 patent, 10:34. “The PE 400 stores local data.” ’273 patent, 10:36. The “memory for the local data” includes “Registers” and “Constants.” ’273 patent, 10:36-54 (“The PE 400 stores local data. The amount of

memory for the local data varies significantly from design to design. ... a design may provide more Constants than Registers. For instance, this may be the case with digital embodiments that use single transistor cells for the Constants (e.g., floating gate Flash memory cells) and multiple transistor cells for the Registers (e.g., 6-transistor SRAM cells). ... Some designs, for instance, may have Register storage but no Constant storage.”), Fig. 4.

211. In addition, because “[e]ach PE needs to operate on its local data,” “within the PE 400 there are data paths 402 a-i, routing mechanisms (such as the *multiplexor MUX 404*), and *components to perform some collection of logical and arithmetic operations* (such as the logic unit 406 and the *LPHDR arithmetic unit 408*).” ’273 patent, 10:58-63. As shown in Figure 4, the “data paths” include data paths 402b and 402d, which both carry data from “REGISTERS” to the “LPHDR ARITH UNIT” 408 (in the case of 402b, the data from “REGISTERS” is one of the possible data items that MUX 404 will route to the LPHDR ARITH UNIT).

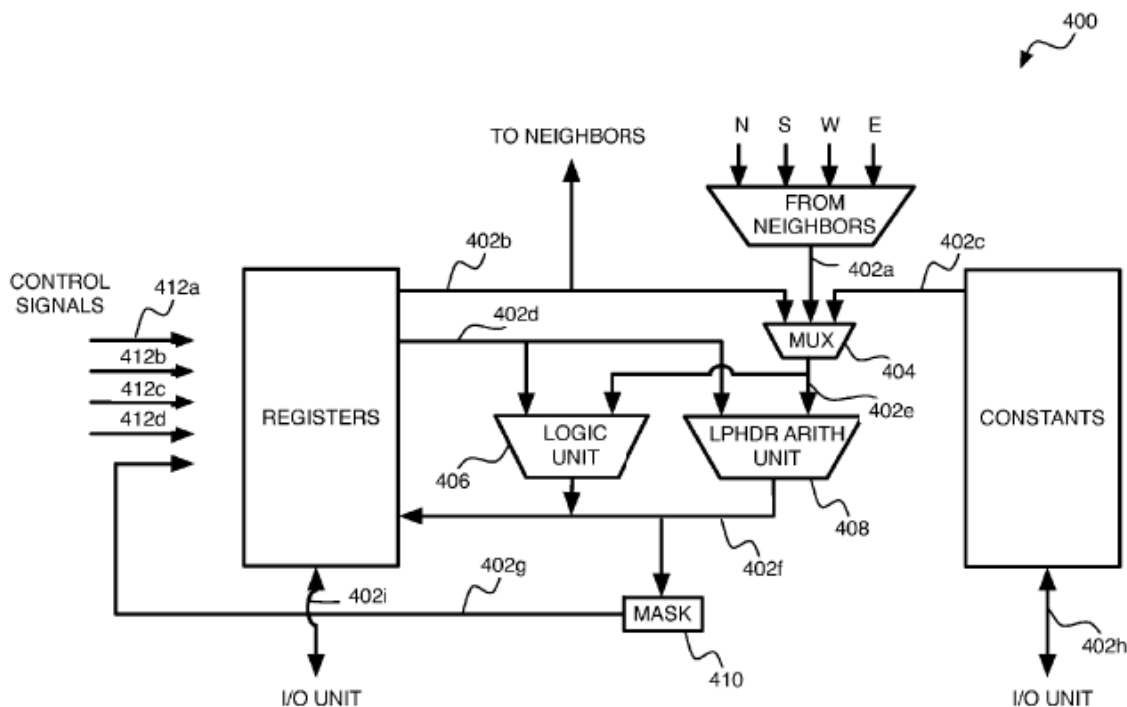


FIG. 4

212. The '273 patent explains that “[t]he operation of the PEs is controlled by control signals 412a-d received from the CU [control unit] 106,” which “specify which *Register* or Constant *memory values* in the PE 400 or one of its neighbors *to send to the data paths, which operations should be performed by the Logic 406 or Arithmetic 408* or other processing mechanisms, where the results should be stored in the Registers, how to set, reset, and use the Mask 410, and so on.” ’273 patent, 11:18-27. Additionally, when describing Figure 6, which “shows an example digital implementation of the LPHDR arithmetic unit 408,” the patent explains that the “unit 408 is controlled by control signals 412a-d, coming from the CU 106, that

determine which available arithmetic operation will be performed on the inputs 602a-b.” ’273 patent, 12:50-51, 12:59-62.

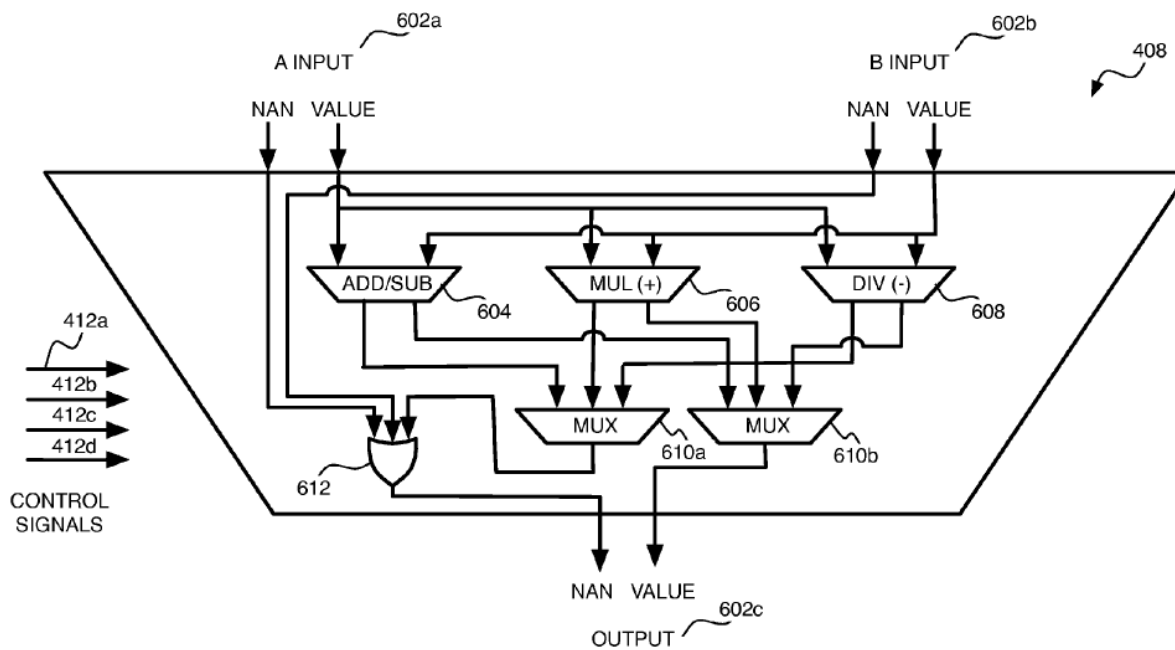


FIG. 6

213. Thus, based on the disclosures in the patent that I discussed in paragraphs 210-212 above, a POSA would have understood that the “processing element” (PE) that the ’273 patent describes as one possible embodiment of an “execution unit” (’273 patent, 8:7-11) pairs registers that store local data with an arithmetic unit that performs arithmetic operations on those data. Dockser’s FPP is similar, as I explain in paragraphs 214-215 below.

214. Like the ’273 patent’s PE, Dockser’s FPP includes registers that store data, in the form of operands. Dockser’s “floating-point processor (FPP) 100,”

which is shown in Figure 1, “includes a floating-point *register file* (FPR) 110; a floating-point controller (CTL) 130; and a floating-point mathematical operator (FPO) 140.” Dockser, [0015], Figure 1. “[T]he floating-point register file 110 includes several addressable register locations 115-1 (REG1), 115-2 (REG2), . . . 115-N (REGN), each configured to *store an operand* for a floating-point operation. The operands may include, for example, data from a memory and/or the results of previous floating-point operations.” Dockser, [0016]. “Each register location 200” in the register file (*see* Dockser, Figure 2) “is configured to store a 32-bit binary floating-point number, in an IEEE-754 32-bit single format.” Dockser, [0017], Figure 2.

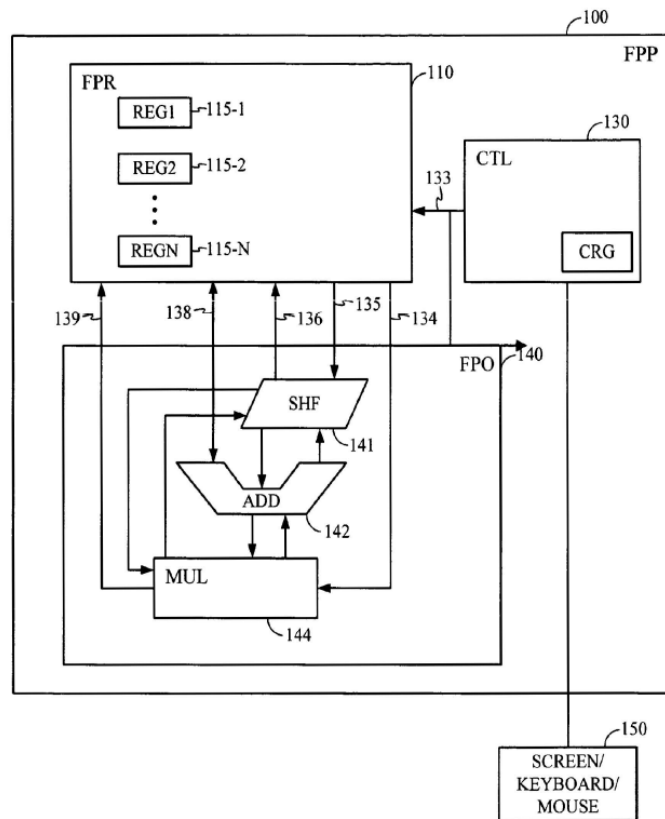


FIG. 1

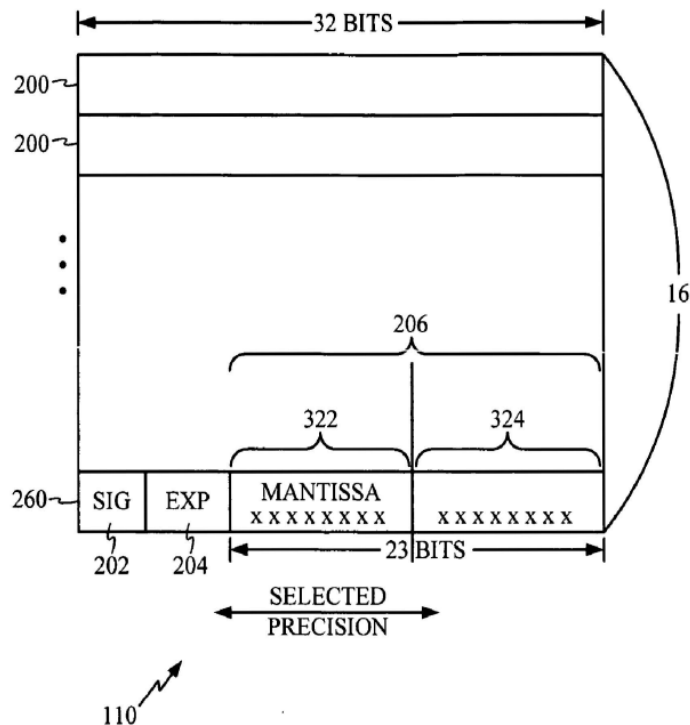


FIG. 2

215. Dockser’s FPP also has an arithmetic unit that performs arithmetic operations on those operands—namely, the “floating-point operator 140,” which “may include one or more components configured to perform the floating-point operations.” Dockser, [0018], [0019]; *see also* Dockser, [0020]-[0022]. “These components may include, but are not limited to, computational units such as a floating-point adder (ADD) 142 configured to execute floating-point add and subtract instructions, and a floating-point multiplier (MUL) 144 configured to execute floating-point multiply instructions.” Dockser, [0019]. “[E]ach of the computational units ADD 142 and MUL 144 in the floating-point operator 140 is

coupled to each other and to the floating-point register file 110 in a way as to allow operands to be transferred between the computational units, as well as between each computational unit and the floating-point register file 110.”

Dockser, [0019]. The coupling of the register file to the operator means that “for each instruction of a requested floating-point operation, the relevant computational unit, i.e. *the adder 142 or the multiplier 144, can receive from the floating-point register file 110 one or more operands* stored in one or more of the register locations.” Dockser, [0023]. “*Upon receiving the operands* from the floating-point register file 110, one or more *computational units in the floating-point operator 140 may execute the instructions of the requested floating-point operation on the received operands*, at the subprecision selected by the floating-point controller 130.” Dockser, [0024]. The subprecision may be specified by “subprecision select bits” written to a “control register” in the FPP. Dockser, [0025].

216. In an alternative mapping, the floating-point operator (FPO) inside Dockser’s FPP is also an “execution unit” as in the ’273 patent’s claims, because the FPO “include[s]... components configured to perform... floating-point operations,” including a floating-point multiplier (MUL) that “execute[s] floating-point multiply instructions.” Dockser, [0019] (“The floating-point operator 140 may include one or more components configured to perform the floating-point

operations. These components may include ... computational units such as ... a floating-point multiplier (MUL) 144 configured to execute floating-point multiply instructions.”). When I refer in this declaration to “Dockser’s execution unit” meeting a claim limitation, I am referring to both Dockser’s FPP and FPO meeting the limitation in the same way because the FPP includes the FPO.

217. A POSA would have understood that Dockser’s FPP (including its FPO) is “low precision” as claimed, because Dockser teaches “the precision” of operations in the FPP is “reduced” (Dockser, e.g., [0014]), and also because Dockser teaches implementing the FPP within the precision range specified in limitation [1B2], as I discuss in Section VI.B.4 below. As I explain in the following paragraph below, the specification of the ’273 patent uses language very similar to the language of limitation [1B2] in characterizing “various possible degrees of precision” (’273 patent, 27:5) that qualify as “low precision;” therefore a POSA would have understood that an execution unit (like Dockser’s) that meets limitation [1B2] is “low precision” as recited in limitation [1A1].

218. The specification states that “[t]he degree of precision of a ‘low precision, high dynamic range’ arithmetic element may vary from implementation to implementation.” ’273 patent, 26:50-52. “For example, in certain embodiments, a LPHDR arithmetic element produces results which are sometimes (or all of the time) no closer than 0.05% to the correct result (that is, the absolute

value of the difference between the produced result and the correct result is no more than one-twentieth of one percent of the absolute value of the correct result)... As another example, a LPHDR arithmetic element may produce results which are sometimes (or all of the time) no closer than 0.2% to the correct result.” ’273 patent, 26:52-27:65. The specification also states that “[t]he frequency with which LPHDR arithmetic elements may yield only approximations to correct results may vary from implementation to implementation.” ’273 patent, 27:29-31.

The specification describes an example where

LPHDR arithmetic elements can perform one or more operations (perhaps including, for example, trigonometric functions), and for each operation the LPHDR elements each accept a set of inputs drawn from a range of valid values, and for each specific set of input values the LPHDR elements each produce one or more output values (for example, simultaneously computing both sin and cos of an input), and the output values produced for a specific set of inputs may be deterministic or non-deterministic. In such an example embodiment, consider further a ***fraction F of the valid inputs and a relative error amount E by which the result calculated by an LPHDR element may differ from the mathematically correct result.*** In certain embodiments of the present invention, for each LPHDR arithmetic element, for at least one operation that the LPHDR unit is capable of performing, ***for at least fraction F of the possible valid inputs to that operation,*** for at least one output signal produced by that operation, ***the statistical mean, over repeated execution, of the numerical***

values represented by that output signal of the LPHDR unit, when executing that operation on each of those respective inputs, *differs by at least E from the result of an exact mathematical calculation of the operation on those same input values*, where F is 1% and E is 0.05%.

'273 patent, 27:32-54. The specification also mentions an example where F is "5%" and E is "0.05%." '273 patent, 27:54-60. This language is very similar to the language of limitation [1B2], with the "fraction F of valid inputs" corresponding to the value 'X' recited in the claim, and the "relative error amount E by which the result calculated by an LPHDR element may differ from the mathematically correct result" corresponding to the value 'Y' recited in the claim. See '273 patent, 30:6-16 (limitation [1B2]) (*"for at least X=5% of the possible valid inputs to the first operation, the statistical mean, over repeated execution of the first operation on each specific input from the at least X% of the possible valid inputs to the first operation, of the numerical values represented by the first output signal of the LPHDR unit executing the first operation on that input differs by at least Y=0.05% from the result of an exact mathematical calculation of the first operation on the numerical values of that same input"*).

219. A POSA would also have understood that Dockser's FPP (including its FPO) is "high dynamic range" as limitation [1A1] recites. The '273 patent describes a 6-bit floating-point exponent as "provid[ing] the desired high dynamic

range.” ’273 patent, 14:53-61 (one embodiment “represents values as low precision, normalized, base 2 floating point numbers ... The exponent may be 6 bits long, or whatever is needed to provide the desired high dynamic range”). Dockser’s FPP (including its FPO) is “high dynamic range” because it uses a standard floating-point representation with an 8-bit exponent (Dockser, [0017]) that provides an even higher dynamic range than the 6-bit exponent that the ’273 patent characterizes as “high dynamic range,” and also because Dockser’s FPP (including its FPO) meets the dynamic range in limitation [1B1], as I explain in Section VI.B.3 below. *See* ’273 patent, 27:5-28 (“implementations may vary in the dynamic range of the space of values they process,” and “a LPHDR arithmetic element [that] processes values in a space which may range approximately from one sixty five thousandth to sixty five thousand” is an “embodiment[]”).

2. [1A2] Adapted to execute a first operation on a first input signal representing a first numerical value to produce a first output signal representing a second numerical value

220. Dockser’s FPP is “adapted to execute” a “first operation” as claimed because it “perform[s]... mathematical *operations*, e.g., multiplication.” *See* Dockser, [0001] (“Floating-point processors are specialized computing units that perform certain mathematical operations, e.g., multiplication...”), [0004] (“An aspect of a method of performing a floating-point operation with a floating-point processor having a precision format is disclosed. The method includes selecting a

subprecision for the floating-point operation...”), [0005] (“One aspect of a floating-point processor having a precision format is disclosed. The floating-point processor includes a floating-point controller configured to select a subprecision for a floating-point operation...”), [0006] (“The floating-point processor includes a floating-point register having a plurality of storage elements configured to store a plurality of floating-point numbers, and a floating-point operator configured to perform a floating-point operation on the one or more of the floating-point numbers stored in the floating-point register.”), [0007] (“The floating-point processor further includes a floating-point controller configured to select a subprecision for a floating-point operation...”), [0008] (“It should be understood that other embodiments of the floating-point processor, and of the method of performing floating-point operations, will become readily apparent to those skilled in the art from the following detailed description, in which various embodiments of the floating-point processor and of the method of performing floating-point operations are shown...”), [0019] (FPP components, including “[t]he floating-point operator 140 may include one or more components *configured to perform the floating-point operations. These components may include*, but are not limited to, computational units such as a floating-point adder (ADD) 142 configured to execute floating-point add and subtract instructions, and *a floating-point multiplier (MUL) 144 configured to execute floating-point multiply*

instructions.”), [0024] (“Upon receiving the operands...one or more computational units in the floating-point operator 140 may *execute the instructions of the requested floating-point operation* on the received operands, at the subprecision selected by the floating-point controller 130.”).

221. The FPP executes the operations at reduced precision on “operands.” *See* Dockser, [0024] (“Upon receiving the operands...one or more computational units in the floating-point operator 140 may *execute the instructions of the requested floating-point operation on the received operands*, at the subprecision selected by the floating-point controller 130.”). These operands are “32-bit binary floating-point *number[s]*.” Dockser, [0016] (“[T]he floating-point register file 110 includes several addressable register locations 115-1 (REG1), 115-2 (REG2), . . . 115-N (REGN), each configured to store an operand for a floating-point operation.”), [0017] (“Each register location 200 is configured to store a 32-bit binary floating-point number, in an IEEE-754 32-bit single format.”).

222. Thus, a POSA would have understood that each operand upon which a mathematical operation is performed by the FPP represents a numerical value, as claimed.

223. The ’273 patent’s “processing element” (PE) 400 (FIG. 4) is one example of an embodiment of an “execution unit... for performing LPHDR

operations.” ’273 patent, 8:7-28, 10:34-36, 2:12-31, 5:65-6:2 (“embodiments... use... (LPHDR) processing elements to perform computations (such as arithmetic operations)”); *see* my discussion in Section VI.B.1 above. “FIG. 4 shows an example design for a PE [processing element] 400.” ’273 patent, 10:34-36. In the “SUMMARY” section, the ’273 patent states that “[e]mbodiments of the present invention are directed to a processor or other device ... which includes *processing elements designed to perform arithmetic operations ... on numerical values of low precision but high dynamic range (‘LPHDR arithmetic’)*.” ’273 patent, 2:11-18. It also states that “[i]n some embodiments, ‘*low precision’ processing elements* perform arithmetic operations which produce results that frequently differ from exact results by at least 0.1% (one tenth of one percent).” ’273 patent, 2:18-31. In the “DETAILED DESCRIPTION” section, the patent alleges that its approach differs from the prior art because “embodiments of the present invention are directed to computer processors or other devices which use *low precision high dynamic range (LPHDR) processing elements* to perform computations (such as arithmetic operations).” ’273 patent, 5:65-6:2. Later, the patent explains that “references herein [*i.e.*, in the patent] to ‘*processing elements*’ within embodiments of the present invention should be understood more generally as any kind of *execution unit, whether for performing LPHDR operations* or otherwise.” ’273 patent, 8:7-11. The patent also explains that in the exemplary system shown

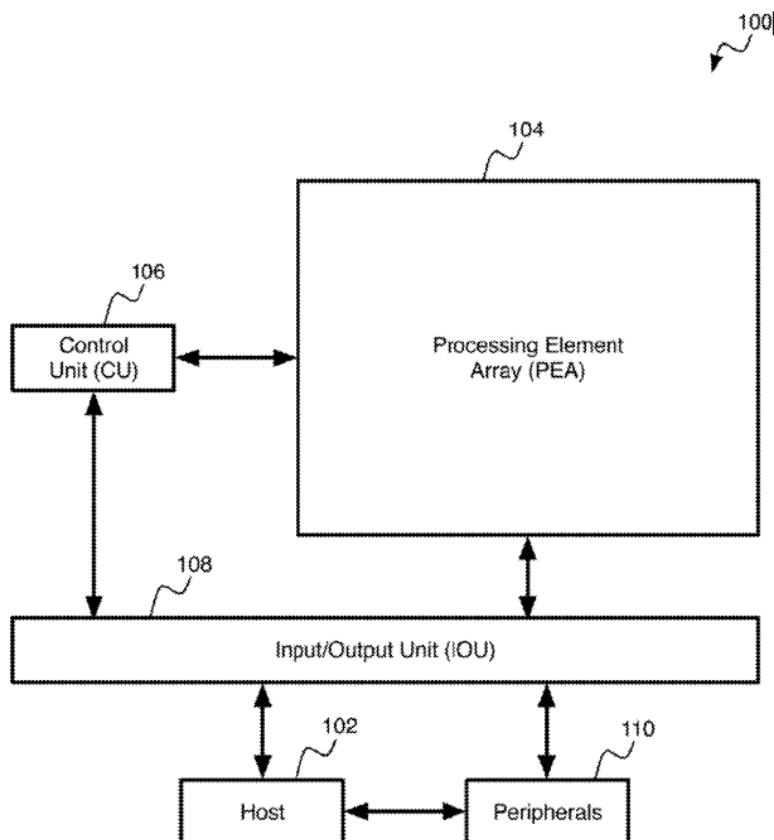
in Figure 1 containing multiple PEs, “one or more (e.g., all) of the PEs ... are LPHDR elements, as that term is used herein.” ’273 patent, 8:25-28.

224. The ’273 patent’s PE 400 receives input via an “I/O [Input/Output] Unit” and “stores” the input values in “Registers” as “local data” on which the PE “operate[s]” and “perform[s] computations.” ’273 patent, 9:6-16, 10:34-57, 8:37-40, FIGs. 1, 4. The patent explains that in the system shown in Figure 1, which includes a “[h]ost,” “[a] goal of the Host 102 is to have the PEA 104 perform massive amounts of computation in a useful way. It does this by causing the PEs to perform computations, typically on data stored locally in each PE, in parallel with one another.” ’273 patent, 8:37-40. “In order to get data into and out of the ... PEA [processing element array] 104, the *I/O Unit 108* may interface the ... PEA 104 with the Host 102, the Host’s memory,” and various other devices. ’273 patent, 9:6-16. The PEA is composed of multiple processing elements (PEs), each of which stores “local data” in “Registers”:

FIG. 4 shows an example design for a PE 400 (which may be used to implement any one or more of the PEs in the PEA 104). The PE 400 stores local data. The amount of memory for the local data varies significantly from design to design. ... Sometimes rarely changing values (Constants) take less room than frequently changing values (Registers), and a design may provide more Constants than Registers. ... Typical storage capacities might be tens or hundreds of arithmetic values stored in the Registers and Constants in each PE, but these

capacities are adjustable by the designer. Some designs, for instance, may have Register storage but no Constant storage. Some designs may have thousands or even many more values stored in each PE. All of these variations may be reflected in embodiments of the present invention.

'273 patent, 10:34-57. In addition, the patent explains that "[e]ach PE needs to operate on its local data." '273 patent, 10:58. Thus, a POSA would have understood that the PE receives input values, stores them as "local data" in Registers, and operates on those values.



'273 patent, Figure 1

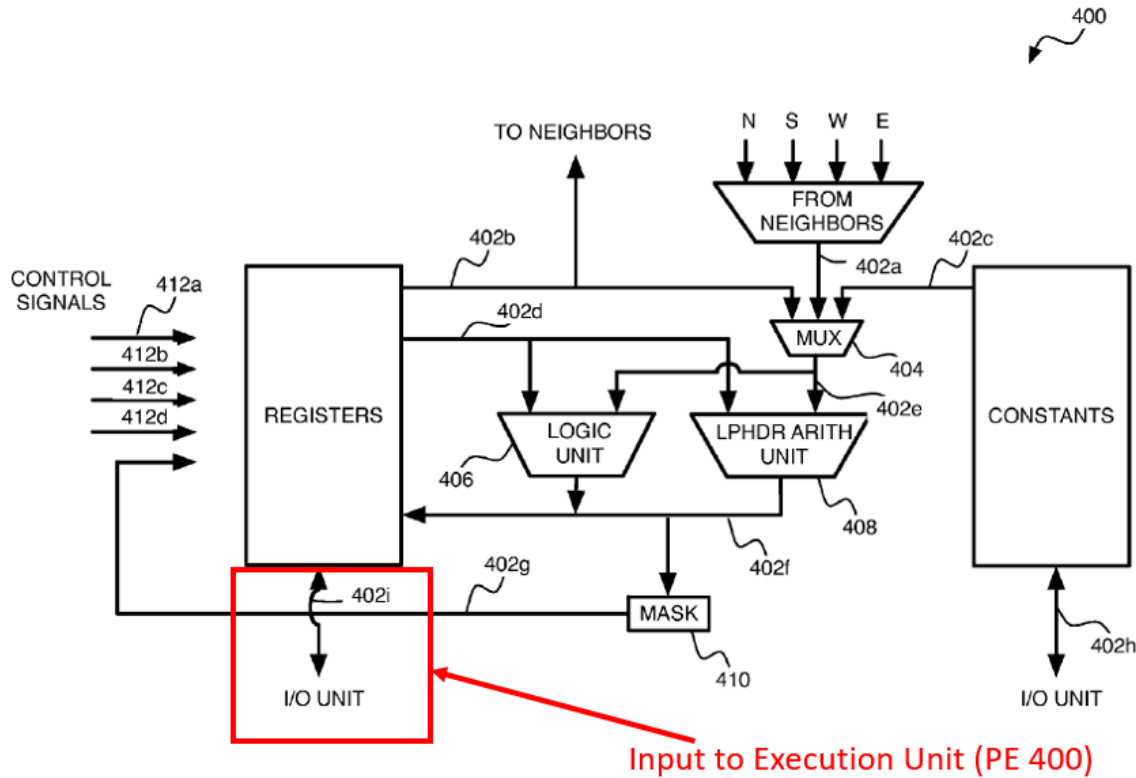


FIG. 4

'273 patent, Figure 4 (annotated)

225. “The input... received by... the PE 400 may... take the form of electrical signals representing numerical values.” ’273 patent, 10:64-67 (“The input, output, and intermediate ‘values’ received by, output by, and operated on by the PE 400 may, for example, take the form of electrical signals representing numerical values.”).

226. The PE “operate[s] on its local data” (*e.g.*, input values received at the Registers) via “data paths 402*a-i*” through “routing mechanisms... and components” that represent values as electrical signals. ’273 patent, 10:58-67. The ’273 patent explains that because “[e]ach PE needs to operate on its local

data,” “within the PE 400 there are data paths 402a-i, routing mechanisms (such as the *multiplexor MUX 404*), and *components to perform some collection of logical and arithmetic operations* (such as the logic unit 406 and the *LPHDR arithmetic unit 408*).” ’273 patent, 10:58-63. As shown in Figure 4, the “data paths” include data paths 402b and 402d, which both carry data from “REGISTERS” to the “LPHDR ARITH UNIT” 408 (in the case of 402b, the data from “REGISTERS” is one of the possible data items that MUX 404 will route to the LPHDR ARITH UNIT). The patent states that “[t]he input, output, and intermediate ‘values’ received by, output by, and operated on by the PE 400 may, for example, take the form of electrical signals representing numerical values”; thus, a POSA would have understood that “routing mechanisms” and “components” in the PE also represent values as electrical signals. ’273 patent, 10:58-67.

227. Based on this description, a POSA would have understood that the claimed execution unit’s “first input signal representing a first numerical value” encompasses an input electrical signal (*e.g.*, 402i in FIG. 4) input to PE 400 (an example of a claimed “execution unit”) at a register as data on which the PE performs an operation via the collective “data paths... and components” of the PE. ’273 patent, 10:58-67.

228. As I explain below, like the ’273 patent’s example PE embodiment, Dockser’s FPP receives input values (operands) at registers, and performs

operations on these inputs via the FPP's collective data paths (e.g., 134-139) and components (e.g., 140-144) in Dockser's FIG. 1.

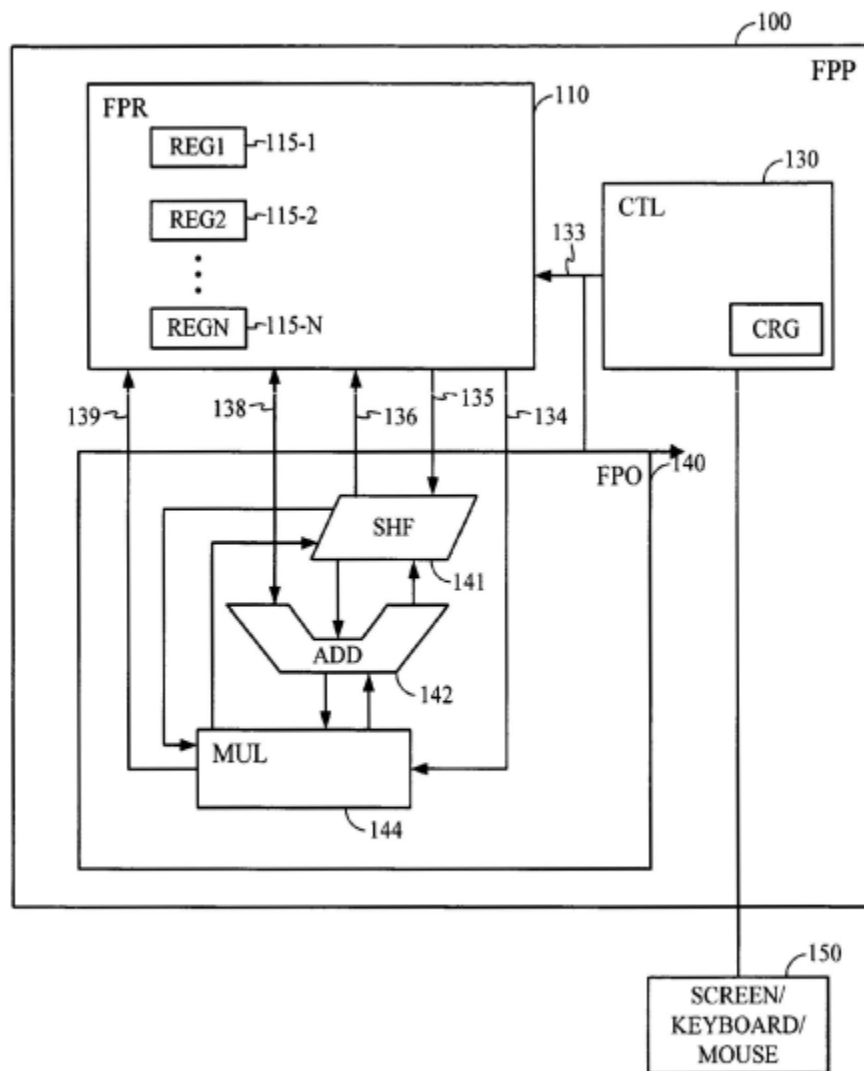


FIG. 1

229. Operands are “move[d]” into Dockser’s FPP’s registers as “data from [the computer system’s main] memory” in 32-bit format. Dockser, [0016] (“[T]he floating-point register file 110 includes several addressable *register locations* ... each configured to *store an operand* for a floating-point operation. The *operands*

may include, for example, *data from a memory* and/or the results of previous floating-point operations. *Instructions* provided to the floating-point processor *may be used to move the operands to and from the main memory.*”), [0017] (“[e]ach *register location 200 is configured to store* a 32-bit binary floating-point *number*, in an IEEE-754 32-bit single format”). A POSA would have understood Dockser thus to describe that the operand “data” being input to the FPP is a “first input signal representing a first numerical value” (*i.e.*, representing the operand “number”—Dockser, [0017]) as claimed, because computing circuits like Dockser’s “move” data between components (*e.g.*, from memory to processor) via electrical signals. For evidence corroborating that this was background knowledge for a POSA, *see, e.g.*, Begun (Ex. 1014), 1:36-37 (“In computer systems, the components communicate via electrical signals.”).

230. Alternatively, to the extent Dockser is not considered to expressly disclose that the data is moved to the FPP via electrical signals, that would have been the straightforward and obvious way to implement what Dockser describes. Moving data via electrical signals was the customary way to move data in computer systems, *see, e.g.*, Begun, 1:36-37, and using this technique to move data to the FPP would have had the predictable and desirable result of having the data reach the FPP in the conventional way.

231. A POSA would thus have understood that Dockser’s FPP is “adapted to execute a first operation [*e.g.*, reduced-precision multiplication] on a first input signal representing a first numerical value [operand]” as recited in limitation [1A2].

232. The FPP’s execution of the operation (*e.g.*, reduced-precision multiplication) on the operands produces an “***output value***” (Dockser also calls it an “***output number***”) represented by “output bits.” Dockser, [0034]. This output value/number is a “second numerical value” as recited in limitation [1A2]. When describing how its multiplier operates at reduced precision, Dockser states:

The ***output value***, resulting from the floating-point multiplication described above, has a width (i.e. number of bits) that is equal to the sum of the widths of the two input values 402 and 404 that are being multiplied together. ***The output value 430 may be truncated to the selected subprecision***, i.e. any of the bits of the output value 430 that are in less the selected precision may be truncated, to generate a truncated ***output number*** characterized by the selected precision. Alternatively, the output value 430 may be rounded off to the selected precision. In either case the ***output bits*** less significant than the selected precision may also be unpowered.

Dockser, [0034]. The “floating-point multiplication described above” referenced in paragraph [0034] refers to the description in paragraphs [0031]-[0034] of how the multiplier operates at reduced precision.

233. A POSA would have understood that the “output value” referred to in paragraph [0034] includes the output of the mantissa multiplication component of the multiplier, and that this mantissa product is combined with the sign bit and 8 exponent bits of the multiplication product to create a 32-bit floating-point output for the multiplier, which is then “sent back to the floating-point register 110 for storage, as shown in FIG. 1.” Dockser, [0024]. A POSA would have understood this for two reasons.

234. *First*, Dockser states that its multiplier is “a conventional floating-point multiplier, configured to perform floating-point multiplication” (Dockser, [0020]) and also states that its multiplier includes “one or more well-known conventional subunits” (Dockser, [0022]). A POSA would have understood that “conventional... floating-point multiplication” involved **adding** the exponents of the two operands together and **multiplying** their mantissas, as the ’273 patent itself acknowledges. *See, e.g.*, ’273 patent, 14:62-65 (“To multiply values” according to “traditional floating-point methods,” “exponents are summed” and “mantissas are multiplied”). Thus, a POSA would have understood that the “multiplication” discussed in paragraphs [0030]-[0034] of Dockser involves reduced-precision multiplication of the mantissa bits of the two operands, and that the bits representing the output of that mantissa multiplication are combined with the exponent sum (and the appropriate sign bit) to form the floating-point output.

235. **Second**, as I noted above, Dockser teaches sending the “output” of floating-point operations “back to floating-point register 110 for storage.” Dockser, [0024]. The “floating-point register file 110” has “register locations” that are “configured to store a 32-bit binary floating-point number, in an IEEE-754 32-bit single format,” which includes both mantissa bits and exponent bits. Dockser, [0017]. Thus, a POSA would have understood that the “output” from Dockser’s “floating-point multiplier (MUL) 144” (Dockser, [0019]) is in the IEEE single format.

236. The output value from Dockser’s multiplication is sent to a register (Dockser, [0024]) from which the “results of previous floating-point operations” (*i.e.*, output values) can be “move[d]... to... the main memory” (Dockser, [0016]). Dockser, [0016] (“[T]he floating-point register file 110 includes several addressable register locations ... each configured to store an operand for a floating-point operation. The *operands may include*, for example, data from a memory and/or *the results of previous floating-point operations*. Instructions provided to the floating-point processor *may be used to move the operands to and from the main memory.*”), (“[0024] (“Upon receiving the operands from the floating-point register file 110, *one or more computational units in the floating-point operator 140 may execute* the instructions of *the requested floating-point operation* on the received operands, at the subprecision selected by the floating-point controller 130.

The output may be sent back to the floating-point register 110 for storage, as shown in FIG. 1.”).

237. A POSA would have understood Dockser thus to describe that the output value sent as bits from the FPP’s registers to the main memory is a “first output signal representing a second numerical value” as recited in limitation [1A2], because the way circuits like Dockser’s move data between such components (e.g., from a processor’s registers to memory) was via electrical signals. *See* Begun, 1:36-37 (“In computer systems, the components communicate via electrical signals.”). Alternatively, to the extent Dockser is not considered to expressly disclose that the data is moved from the FPP to memory via electrical signals, that would have been the straightforward and obvious way to implement what Dockser describes. Moving data via electrical signals was the customary way to move data in computer systems, *see, e.g.*, Begun, 1:36-37, and using this technique to move data from the FPP to memory would have had the predictable and desirable result of having the data reach the memory in the conventional way.

238. A POSA would thus have understood that Dockser’s FPP is “adapted to execute [the] first operation [*e.g.*, reduced-precision multiplication]... to produce a first output signal [electrical signal sending output bits to memory] representing a second numerical value [the output number from the

multiplication]” as limitation [1A2] recites, as shown in my annotated version of Dockser’s Figure 1 below.

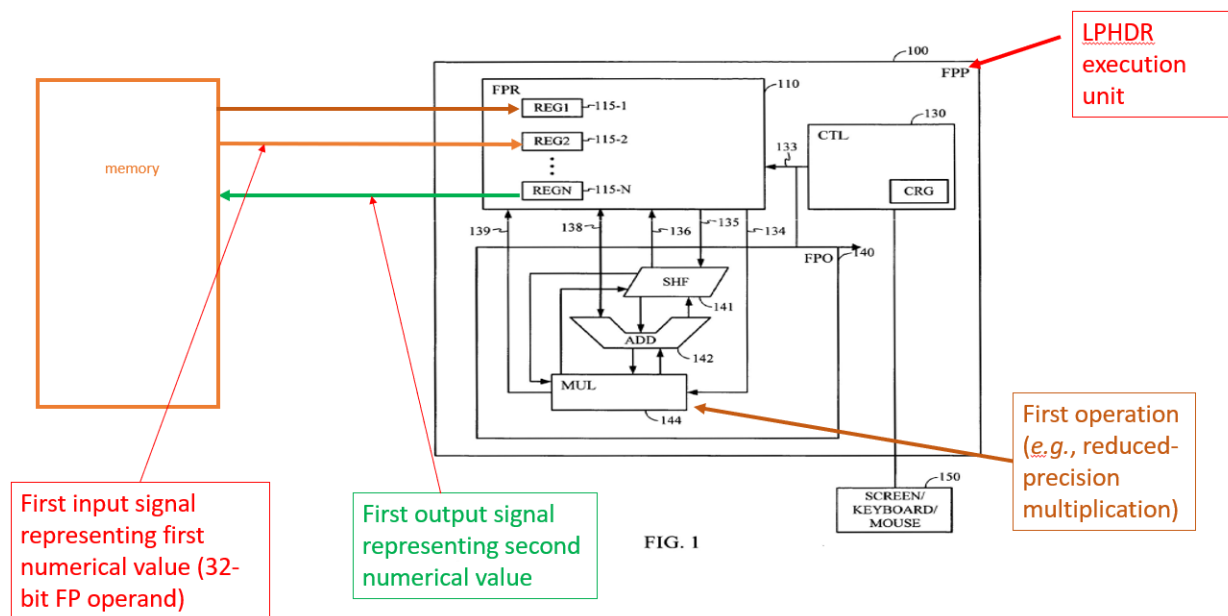


FIG. 1

Dockser, Figure 1 (annotated)

239. In the alternative mapping where Dockser’s FPO meets the claimed “execution unit” (*see* Section VI.B.1 above, paragraph 216), the FPO is “adapted to execute [the] first operation [reduced-precision multiplication] on a first input signal representing a first numerical value [signal 134 inputting operands to FPO] to produce a first output signal representing a second numerical value [signal 139 outputting multiplication result from FPO]” as limitation [1A2] recites, as shown in my annotated version of Dockser’s Figure 1 below.

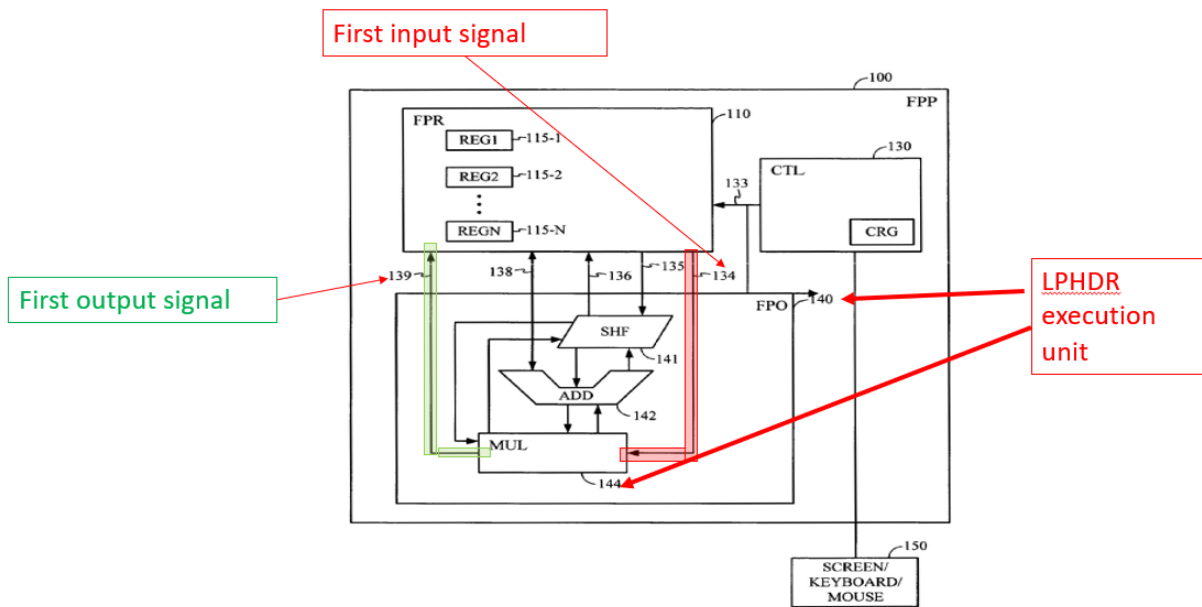


FIG. 1

Dockser, Figure 1 (annotated)

3. [1B1] wherein the dynamic range of the possible valid inputs to the first operation is at least as wide as from 1/65,000 through 65,000

240. As I explained in Sections VI.B.1-VI.B.2 above (discussing Dockser, [0016]-[0017], [0024]), Dockser's FPP performs a first operation (*e.g.*, reduced-precision multiplication) on operands that are IEEE-754 32-bit single-format numbers having 8-bit exponents. (*See* Dockser, [0002]). As was well-known in the art, the number of exponent bits determines the dynamic range in a floating-point representation (including the IEEE-754 single format). For evidence corroborating that this was background knowledge for a POSA, *see, e.g.*, Hekstra (Ex. 1013), 4:47-52 ("A definition of floating-point numbers is given. A floating-

point number x is represented by a tuple (e_x, m_x) with: exponent e_x . The number of bits N_e determine the dynamic range of the representation. [M]antissa m_x . The number of bits N_m , not including the sign bit, determine the precision of the representation.”); Gaffar (Ex. 1012), page 4 (“In the case of floating-point ... the range depends on the exponent bit-width.”); Grinberg (Ex. 1015), 8:48-49 (“The dynamic range increases with the number of bits dedicated to the exponent...”).

241. IEEE-754 8-bit exponents range from -126 to 127; thus the dynamic range of “normal” IEEE-754 single-format operands is from around 2^{-126} (massively smaller than 1/65,000) to around 2^{127} (massively larger than 65,000). For evidence corroborating that this was background knowledge for a POSA, *see, e.g.,* Dockser-Lall, [0004] (“In the IEEE 754 standard, the value of the exponent for a single-precision floating-point number ranges from -126 to 127.”); Tong, page 274 (“... an IEEE single-precision number ... require[s] 32 bits of storage ... the dynamic range provided by the FP representation ($1.99999988 \times 2^{-126}$ to $1.99999988 \times 2^{127}$)”).

242. A POSA would have understood Dockser to disclose that its FPP supports the “normal” IEEE-754 single-format numbers as operands, as Dockser makes no reference to supporting the far less common “denormal” numbers (smaller than 2^{-126}), let alone to supporting denormal numbers exclusively. For

evidence corroborating the POSA's background knowledge in this regard, *see, e.g.*, Dockser-Lall, [0005], [0008] ("In common processor applications...denormal numbers need not always be supported."). To the extent Dockser is not considered to expressly disclose that the FPP operates on "normal" IEEE-754 single-format numbers as operands, that would have been a conventional and obvious way to implement what Dockser discloses. Not supporting "denormal" floating-point numbers was one of two choices (supporting them or not supporting them) when supporting IEEE-754 single-format numbers as Dockser discloses, and not supporting "denormal" numbers was the more common choice because hardware support for numbers smaller than the "normal" number range required special circuits, and POSAs knew that "such additional circuits increase silicon area, increase latency, and introduce throughput delay, potentially increasing the minimum cycle time and hence reducing the maximum operating frequency. Additionally, denormal numbers are rarely encountered, and optimizing performance for the rare case at the expense of the common case reduces overall processor performance." Dockser-Lall, [0010].

243. Thus "the dynamic range of the possible valid inputs" (normal IEEE-754 single-format operands) to Dockser's FPP operation "is at least as wide as from 1/65,000 through 65,000," as recited in limitation [1B1].

244. As I discuss below (in Section VI.B.4), Dockser’s precision-reduction techniques within the FPP do not change the exponents of operated-on numbers (they only reduce precision in the mantissas); thus the dynamic range of the possible valid inputs in the alternative mapping where the FPO meets the claimed “execution unit” (*see* Section VI.B.1 above, paragraph 216) is the same as that of the FPP inputs.

4. **[1B2] for at least X=5% of the possible valid inputs to the first operation, the statistical mean, over repeated execution of the first operation on each specific input from the at least X% of the possible valid inputs to the first operation, of the numerical values represented by the first output signal of the LPHDR unit executing the first operation on that input differs by at least Y=0.05% from the result of an exact mathematical calculation of the first operation on the numerical values of that same input;**

- a. **The “Statistical Mean” Limitation**

245. The ’273 patent’s specification’s only mention of “the statistical mean, over repeated execution of... [the] operation on each... respective input[.]” (’273 patent, 27:44-62) is in the context of referencing “non-deterministic” embodiments. *See* ’273 patent, 27:31-44 (“LPHDR arithmetic elements can perform one or more operations” where “the LPHDR elements each accept a set of inputs...and for each specific set of input values ... the output values produced for a specific set of inputs may be deterministic or *non-deterministic*.”).

246. A POSA would have understood the claims’ “statistical mean” limitation in the context of the ’273 patent’s stated intent to claim not only “repeatable” deterministic embodiments like digital computing circuits that always produces the same output when repeating the same operation on the same input, but also analog embodiments that are non-deterministic because they “introduce noise into their computations, so the computations are not repeatable.” ’273 patent, 4:7-13, 14:16-61, 17:10-14. In discussing the prior art, the patent describes prior art “[a]rray processors” that “analog representations of numbers and analog circuits to perform computations,” and states that the “**SCAMP**” computer “is such a machine.” ’273 patent, 4:7-9. The patent notes that “[t]hese machines...introduce noise into their computations, so the computations are not repeatable.” ’273 patent, 4:12-13. Later, when describing its embodiments, the patent states that “[s]ome embodiments of the present invention may include analog representations and processing methods.” ’273 patent, 14:16-17. The patent states that “[s]uch methods, often called Analog methods, can be used to perform LPHDR arithmetic in the broad range of architectures we have discussed, of which SIMD is one example,” and then states that “[a]n example of an *analog SIMD* architecture is the **SCAMP** design (and related designs) of **Dudek**,” where “values have low dynamic range” and are “accurate roughly to within 1%.” ’273 patent, 14:23-28. The patent later purportedly describes “how to build an analog

SIMD machine that performs LPHDR arithmetic, and is thus an embodiment of the present invention,” ’273 patent, 14:50-52, and states that in this purported machine, “the analog value may be accurate to about 1%, following the approach of *Dudek*,” ’273 patent, 14:57-58. Thus, a POSA would have understood that the patent’s “analog” embodiment is based on the SCAMP design by Dudek, and hence that the analog embodiment is non-deterministic because “[its] computations are not repeatable.” ’273 patent, 4:13. In contrast, when describing the “embodiment [that] uses logarithmic arithmetic,” the patent states that “[t]he arithmetic is *repeatable*, that is, *not* noisy.” ’273 patent, 17:10-14.

247. For example, if an analog circuit represents the number 1 as a voltage somewhere between 0.99 and 1.01 (’273 patent, 14:13-30), then the output of a first execution of an addition operation that adds 1+1 may be 1.98, a second execution of the same operation may produce a result of 2.01, and repeated executions may produce different results unpredictably. Such a circuit would be as described in the ’273 patent at 14:13-30, which describes a circuit that “may...represent LPHDR values as...voltages” (14:18-19) and may be “accurate to roughly within 1%” (14:28-30).

248. A POSA would have understood that for such non-deterministic embodiments, the claimed “statistical mean, over repeated execution of the first operation on each specific input..., of the numerical values represented by the first

output signal of the LPHDR unit executing the first operation on that input”

limitation in [1B2] refers to averaging (taking the statistical mean of) the different outputs produced by the same operation on the same input—*e.g.*, the mean of 1.98 and 2.01 is 1.995 in the above example.

249. On the other hand, for deterministic digital embodiments, a POSA would have understood that the claimed “statistical mean, over repeated execution of the first operation on each specific input from the at least X % of the possible valid inputs to the first operation, of the numerical values represented by the first output signal of the LPHDR unit executing the first operation on that input” is the same as the numerical value of the first output signal for any individual execution of the first operation on each specific input, because that output is always the same for any specific input.

250. Dockser’s multiplication operation uses “a conventional floating-point multiplier” (Dockser, [0020]) which a POSA would have understood is a conventional digital circuit that is deterministic. For evidence corroborating that this was background knowledge for a POSA, *see, e.g.*, Weiss (Ex. 1011), 1:40-42 (“Moreover, ***conventional digital circuits*** are traditionally deterministic. Thus, the output of a conventional digital circuit is typically predictable.”). A POSA would have understood that Dockser’s FPP, and the “conventional floating-point multiplier” in the FPP, is a digital circuit because, *e.g.*, it processes “bits,” and it is

a “conventional” floating-point multiplier. *See, e.g.*, Dockser, Abstract, [0004]-[0007], [0020]. *See also* ’273 patent, 15:32-33 (referring to “*traditional*,” *i.e.* conventional, “digital floating point arithmetic”).

251. Therefore, a POSA would have understood that repeatedly executing Dockser’s multiplication operation on the same input (*i.e.*, pair of operands) with the same selected precision level applied will yield the same result for every execution, so that the statistical mean of the outputs is the same as the output for any single execution.

252. To the extent Dockser is not considered to expressly disclose that its “conventional floating-point multiplier” (Dockser, [0020]) is a deterministic digital circuit, that would have been the conventional and obvious way to implement Dockser’s conventional floating-point multiplier. As I explain in paragraph 250 above, implementing binary floating-point multipliers as deterministic digital logic circuits was the customary way to implement such multipliers, and implementing them in this way would have had the predictable and desirable result of allowing the FPP to perform floating-point multiplication in the conventional way.

b. The “Exact Mathematical Calculation” Limitation

253. The numerical value of each input operand to Dockser’s operation (reduced-precision multiplication) is the number represented by the IEEE-754 32-bit single-format bit sequence (Dockser, [0016]-[0018]; *see* my discussion in

Sections VI.B.2-VI.B.3 above), and the claimed “result of an exact mathematical calculation of the first operation on the numerical values of that same input” is the product (which has more than 32 bits) that would result if the numbers represented by the pair of input 32-bit operands were multiplied *without* reducing precision. The numerical value represented by Dockser’s output signal differs from this “exact mathematical calculation” because Dockser performs a reduced-precision multiplication, as I explain in Section VI.B.4.c below.

c. The Relative Error Amount (Y) for a Fraction of the Possible Valid Inputs (X) Limitation

254. Limitation [1B2] recites that “for at least $X=5\%$ of the possible valid inputs to the first operation,” the statistical mean limitation (which is met in Dockser by the numerical value represented by the output signal for any single execution of the first operation on a specific valid input, as I discussed in Section VI.B.4.a above) “differs by at least $Y=0.05\%$ from” the exact mathematical calculation that I discussed in Section VI.B.4.b above. I have illustrated this in the annotated version of limitation [1B2] shown below, where the red text corresponds to the “statistical mean” limitation and the blue text corresponds to the “exact mathematical calculation” limitation.

[1B2] for at least $X=5\%$ of the possible valid inputs to the first operation, **the statistical mean, over repeated execution of the first operation on each specific input from the at least $X\%$ of the possible valid inputs to the first operation, of the numerical values represented**

by the first output signal of the LPHDR unit executing the first operation on that input differs by at least $Y=0.05\%$ from the result of an exact mathematical calculation of the first operation on the numerical values of that same input; and

255. The '273 patent refers to the “Y” percentage in limitation [1B2] as the “relative error amount E by which the result calculated by [the] LPHDR element... differ[s] from the mathematically correct result,” and refers to the the “X” percentage in limitation [1B2] as the “fraction F of the valid inputs” for which the operation produces at least that relative error amount:

In such an example embodiment, consider further *a fraction F of the valid inputs* and a *relative error amount E* by which the result calculated by an LPHDR element may differ from the mathematically correct result. In certain embodiments of the present invention, for each LPHDR arithmetic element, for at least one operation that the LPHDR unit is capable of performing, *for at least fraction F of the possible valid inputs to that operation, for at least one output signal produced by that operation, the statistical mean, over repeated execution, of the numerical values represented by that output signal of the LPHDR unit, when executing that operation on each of those respective inputs, differs by at least E from the result of an exact mathematical calculation of the operation on those same input values*, where F is 1% and E is 0.05%. In several other example embodiments, F is not 1% but instead is one of 2%, or 5%, or 10%, or 20%, or 50%. For each of these example embodiments, each with some specific value for F, there are other example embodiments in

which E is not 0.05% but instead is 0.1%, or 0.2%, or 0.5%, or 1%, or 2%, or 5%, or 10%, or 20%. These varied embodiments are merely examples and do not constitute limitations of the present invention.

'273 patent, 27:40-62.

256. Dockser's multiplication operation receives a pair of (normal) IEEE-754 32-bit single-format operands as input. *See* my discussion in Section VI.B.3 above for limitation [1B1]; Dockser, [0016]-[0017], [0024]-[0027]. Therefore, a POSA would have understood that the claimed "possible valid inputs" in Dockser are the set of possible normal IEEE-754 32-bit single-format numbers forming pairs of operands in input signals to the execution unit that can be multiplied together to produce an output representing a numerical value as claimed (rather than, e.g., an "exception" that indicates an overflow or an underflow). As I explain in Appendix I.A below, a POSA would have understood that pairs of operands that do not produce a numerical value as output when multiplied together are not "possible valid inputs," because they do not produce an output signal representing a numerical value as a result of the operation, as the challenged claims recite.

257. As I explain in the following paragraphs and in subsections VI.B.4.c(1)-VI.B.4.c(3) below, a POSA would have understood that Dockser meets limitation [1B2] because Dockser operates the FPP and FPO at precision levels where the claimed relative error amount (Y) is at least 0.05% for at least X=5% of all valid input pairs of normal IEEE-754 32-bit single-format numbers.

258. Dockser’s FPP (including its FPO) operates at a precision level selected by specifying the number of mantissa fraction bits to be retained, and the remaining “excess bits” are dropped, as I explain in paragraphs 259-262 below. Dockser, [0004]-[0007], [0026]-[0029].

259. Dockser’s FPP operates at a selected precision level. Dockser teaches “a method of performing a floating-point operation with a floating-point processor having a precision format” that includes “*selecting a subprecision* for the floating-point operation on one or more floating-point numbers.” Dockser, [0004]. Dockser also teaches “a floating-point processor having a precision format”; this processor “includes a floating-point controller configured to *select a subprecision* for a floating-point operation on one or more floating-point numbers.” Dockser, [0005]. The precision level is selected by specifying the number of mantissa fraction bits to be retained, with the remaining bits being treated as “excess” bits. *See* Dockser, [0004] (“the selection of the subprecision result[s] in one or more *excess* bits for each of the one or more floating-point numbers”—in other words, when a “*subprecision*” is selected, certain bits become “excess”), [0005] (“*the selection of the subprecision resulting in one or more excess bits* for each of the one or more floating-point numbers”—in other words, when the “controller” selects a “*subprecision*” for the operation, certain bits become “excess”).

260. The bits that are “excess” bits as a result of the precision selection (Dockser, [0004]-[0007]) are dropped by removing power from the components used to store or process them. Dockser, [0004] (Dockser’s method “further includes removing power from one or more *components* in the floating-point processor that would otherwise be used to store or process the one or more excess bits, and *performing the floating-point operation with power removed from the one or more components*”), [0005] (“controller” in floating-point processor is “further configured to *remove power from one or more components* in the floating-point processor that would otherwise be used to *store or process* the one or more excess bits”), [0006] (“The floating-point *processor includes a floating-point register having* a plurality of *storage elements* configured to store a plurality of floating-point numbers, and a floating-point operator configured to perform a floating-point operation on the one or more of the floating-point numbers stored in the floating-point register. The floating-point processor further includes a floating-point controller configured to select a subprecision for a floating-point operation on said one or more of the floating-point numbers, the selection of the *subprecision resulting in one or more excess bits* for each of said one or more of the floating-point numbers, *the one or more excess bits being stored in one or more of the storage elements* of the floating-point register, and wherein the floating-point controller is further configured to *remove power from the storage*

elements for the one or more excess bits”), [0007] (“*A further aspect* of a floating-point processor having a precision format is disclosed. The floating-point processor includes a floating-point register configured to store a plurality of floating-point numbers, and a floating-point operator having logic configured to perform a floating-point operation on the one or more of the floating-point numbers stored in the floating-point register. The floating-point processor further includes a floating-point controller configured to select a subprecision for a floating-point operation on said one or more of the floating-point numbers, *the selection of the subprecision resulting in one or more excess bits* for each of said one or more of the floating-point numbers, and wherein the floating-point controller is further configured to *remove power a portion of the logic that would otherwise be used to process the one or more excess bits.*”), [0025] (“subprecision select bits are written to the control register 137” in the exemplary FPP shown in Figure 1, “which in turn controls the bit length of the mantissa for each operand during the floating-point operation”), [0026] (“The subprecision select bits may be used to reduce the precision of the floating-point operation. This may be achieved in a variety of ways. In one embodiment, the floating-point controller 130 may cause *power to be removed from the floating-point register elements for the excess bits* of the fraction that are not required to meet the precision specified by

the subprecision select bits.’’), [0027] (‘‘In addition, the *logic* in the floating-point operator 140 corresponding to the *excess mantissa bits do not require power.*’’).

261. The bits that are not rendered ‘‘excess’’ by the selection of a subprecision (Dockser, [0004]-[0007], [0026]) are retained by maintaining power to the components that store and process them, as I explain below and in the subsequent paragraph. Figure 2 of Dockser shows a register location (element 260) with a ‘‘1-bit sign 202, an 8-bit exponent 204, and a 24-bit fraction 206.’’ Dockser, [0017]. A POSA would have understood that the reference in paragraph [0017] to a ‘‘24-bit fraction’’ was a typographical error and that it should say ‘‘23-bit fraction,’’ since Dockser elsewhere makes clear (as a POSA would have understood) that only 23 fraction bits are stored out of a 24-bit mantissa. *See* Figure 2 (showing ‘‘23 bits’’ for element 206), [0002] (‘‘Only the 23 fraction bits of the mantissa are stored in the 32-bit encoding, an integer bit, immediately to the left of the binary point, is implied.’’), [0026] (‘‘if each location in the floating-point register file contains a 23-bit fraction, and the subprecision required for the floating-point operation is 10-bits, only the 9 commonly significant bits (MSBs) of the fraction are required; the hidden or integer bit makes the tenth’’). Figure 2 shows the mantissa bits 206 divided into two segments with reference numbers 322 and 324, with segment 324 shown to the right of the ‘‘selected precision’’ line. As Dockser explains when discussing Figure 3A, which refers to those same elements,

element 322 refers to “powered bits,” while element 324 refers to “unpowered bits.” Docker, [0029], Fig. 3A. As shown in Figure 3A, the unpowered bits are the bits to the right of the line with element number 305, which indicates the “selected reduced precision.” Dockser, Fig. 3A; *see also* Dockser, [0029] (“In the example illustrated in FIG. 3A, the selected subprecision is represented with a line 305.”).

262. A POSA would have understood that the “unpowered bits” labeled element 324 in Figures 2 and 3A are the “excess” bits, as shown in the annotated version of Figure 2 below, since they are to the right of the boundary line that indicates the “selected precision” in Figure 2 and to the right of the line labeled “selected reduced precision” in Figure 3. The fact that those bits are to the right of the line means that they are less significant than those to the left, because in Dockser, “[i]n accordance with *standard convention*, the floating-point register stores in order the bits that make up each number, ranging from a rightmost LSB [least significant bit] to a leftmost MSB [most significant bit].” Dockser, [0028]. A POSA would have understood that the “excess” bits are the bits that are *less* significant than the bits corresponding to the selected precision/subprecision. *See, e.g.*, Dockser, [0026] (“The subprecision select bits may be used to reduce the precision of the floating-point operation....In one embodiment, the floating-point controller 130 may cause *power to be removed from* the floating-point register

elements for the *excess bits* of the fraction that are not required to meet the precision specified by the subprecision select bits. By way of example, if each location in the floating-point register file contains a 23-bit fraction, and the subprecision required for the floating-point operation is 10-bits, only the **9 commonly significant bits (MSBs)** of the fraction are required; the hidden or integer bit makes the tenth. *Power can be removed* from the floating-point register elements for the *remaining 14 fraction bits.*”).

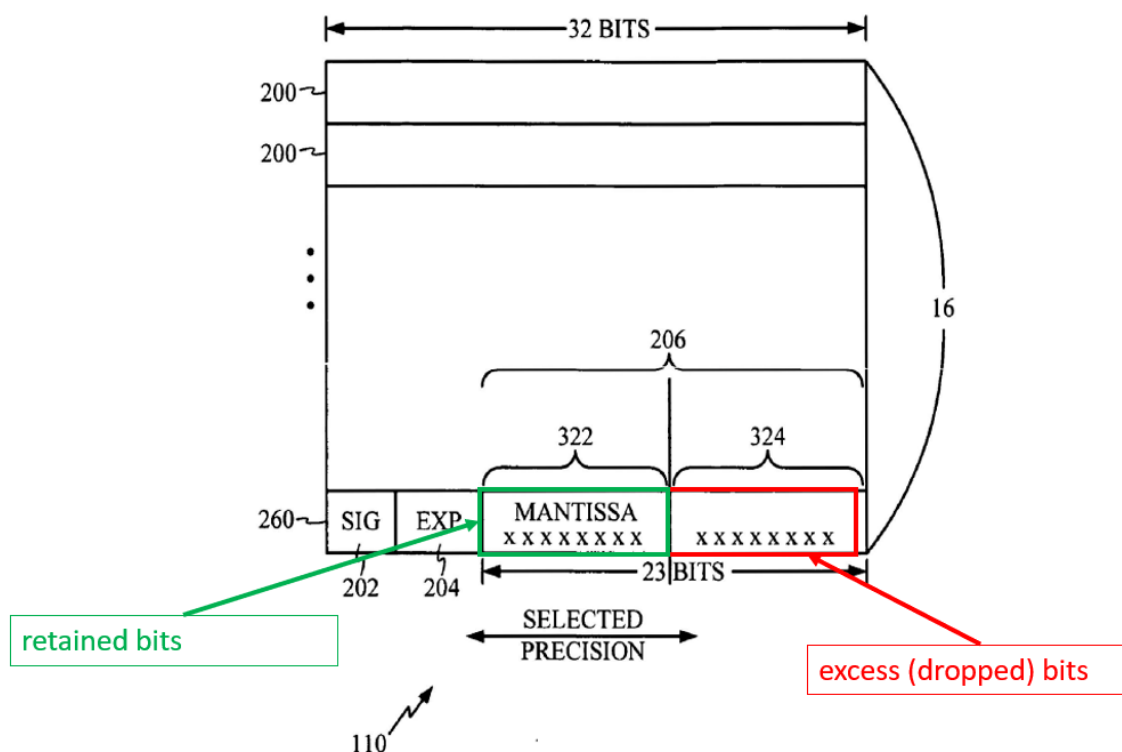


FIG. 2

Dockser, Figure 2 (annotated)

263. In one “example” of selecting a precision level, Dockser retains only the 9 most-significant bits (MSBs)—i.e., the leftmost 9 bits—of the mantissa fraction, and dropping the remaining 14 mantissa bits. Dockser, [0025]-[0028]. In this example, “select bits are written to the control register 137” which is “in the floating-point controller 130.” Dockser, [0025]. The controller “may cause power to be removed from the floating-point register elements for the excess bits of the fraction that are not required to meet the precision specified by the subprecision select bits.” Dockser, [0026]. “By way of example, if each location in the floating-point register file contains a 23-bit fraction, and the subprecision required for the floating-point operation is 10-bits, *only the 9 commonly significant bits (MSBs) of the fraction are required*; the hidden or integer bit makes the tenth. *Power can be removed from the floating-point register elements for the remaining 14 fraction bits.*” Dockser, [0026]. A POSA would have understood that the 9 most significant bits are the 9 *leftmost* bits, because in Dockser, “[i]n accordance with *standard convention*, the floating-point register stores in order the bits that make up each number, ranging from a rightmost LSB [least significant bit] to a leftmost MSB [most significant bit].” Dockser, [0028].

264. Dockser discloses two precision-reducing techniques that can be used separately or together, as I explain in paragraphs 265-269 below. Dockser, [0004]-[0007], claims 9-11.

265. One technique drops bits from the operands by removing power from storage elements in the FPP's registers that correspond to the excess mantissa bits, which are dropped. *See* Dockser, [0006] (“The floating-point processor further includes a floating-point controller configured to select a subprecision for a floating-point operation on said one or more of the floating-point numbers, the selection of the subprecision resulting in one or more excess bits for each of said one or more of the floating-point numbers, the one or more *excess bits being stored in one or more of the storage elements of the floating-point register*, and wherein the floating-point controller is further configured to *remove power from the storage elements for the one or more excess bits.*”), [0026] (“In one embodiment, the floating-point controller 130 may *cause power to be removed from the floating-point register elements for the excess bits* of the fraction that are not required to meet the precision specified by the subprecision select bits.”). As indicated in the annotated version of Figure 1 below, this technique involves removing power from storage elements in the registers in the register file.

266. The other technique drops bits by removing power from elements within the multiplier logic that computes the product of the operand mantissas. *See* Dockser, [0007] (“the selection of the subprecision resulting in one or more excess bits for each of said one or more of the floating-point numbers, and wherein the floating-point controller is further configured to *remove power a portion of the*

logic that would otherwise be used to process the one or more excess bits.”). See also Dockser, [0027] (“the logic in the floating-point operator 140 *corresponding to the excess mantissa bits do not require power*. Thus, power savings may be achieved by *removing power to the logic* in the floating-point operator 140 that remains unused as a result of the subprecision selected.”). This technique involves removing power in parts of the floating-point operator 140, as shown in the annotated version of Figure 1 below. Dockser describes in paragraphs [0030]-[0034], which I will discuss in Section VI.B.4.c(2) below, how this technique works in the case where the “logic” includes logic in “multiplier MUL,” which is element 144 shown in the floating-point operator 140 in Figure 1.

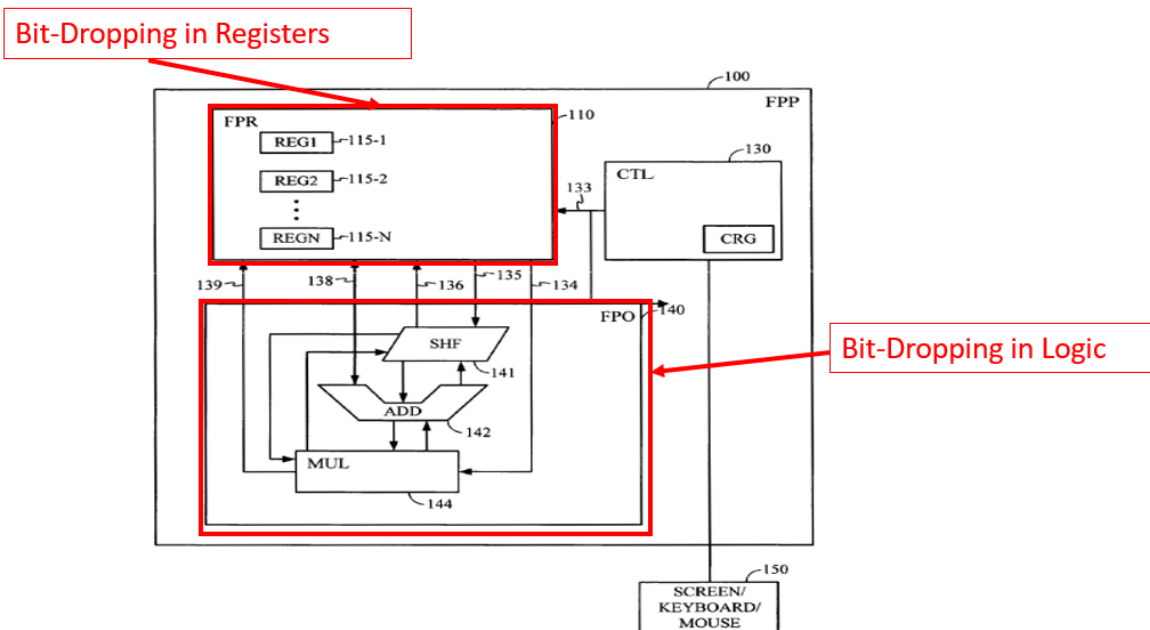


FIG. 1

Dockser, Figure 1 (annotated)

267. A POSA would have understood Dockser to describe that these two techniques can be used individually or together. In paragraph [0005], Dockser states that “the floating-point controller” is “further configured to remove power from *one or more components* in the floating-point processor that would otherwise be used to *store or process* the one or more excess bits.” Dockser, [0005]. In other words, power can be removed from two types of “components” – those used to “store” “excess bits,” and those used to “process” them. Then, in the next two paragraphs, Dockser describes these techniques as two “aspects” of a floating-point processor. *See* Dockser, [0006] (“Another aspect of a floating-point processor having a precision format is disclosed ... the floating-point controller is further configured to remove power from the storage elements for the one or more excess bits.”), [0007] (“A further aspect of a floating-point processor having a precision format is disclosed ... the floating-point controller is further configured to remove power a portion of the logic that would otherwise be used to process the one or more excess bits.”). Thus, a POSA would have understood that Dockser teaches removing power from “storage elements” that store the excess bits, or from “logic that would otherwise be used to process” the excess bits, or both.

268. The claims of Dockser also confirm that these techniques can be used separately or together. Claim 8 of Dockser recites:

8. A floating-point processor having a maximum precision comprising:

a floating-point controller configured to select a subprecision less than the maximum precision for a floating-point operation on one or more floating-point numbers, the selection of the subprecision resulting in one or more excess bits for each of the one or more floating-point numbers, the floating-point controller being further configured to ***remove power from one or more components*** in the floating-point processor that would otherwise be used to store or process the one or more excess bits; and

a floating-point operator configured to perform the floating-point operation.

269. Claim 9, which depends from claim 8, recites that “the one or more ***components from which power can be removed includes the storage elements*** for the one or more excess bits.” Claim 10, which depends from claim 9—and thus includes claim 9’s limitations—recites that “the one or more ***components*** from which power can be removed ***includes a portion of the logic that would otherwise be used to process the one or more excess bits.***” Claim 11 recites this same language from claim 10, but depends directly from claim 8 and therefore ***does not*** include claim 9’s limitations. Thus, based on the claims, a POSA would have understood that the “components” from which Dockser teaches removing power

are “storage elements” (claim 9), “a portion of the logic” (claim 11), or both (claim 10).

270. In addition to teaching register bit-dropping and logic bit-dropping, Dockser states in paragraph [0034] that “the output value 430,” *i.e.* the output value produced by the mantissa multiplication discussed in paragraphs [0030]-[0033] and Figure 3B, “may be truncated to the selected subprecision, *i.e.* any of the bits of the output value 430 that are in less the selected precision may be truncated, to generate a truncated output number characterized by the selected precision,” and that “output bits less significant than the selected precision may also be unpowered.” Dockser, [0034] & Figure 3B (reproduced below, annotated). A POSA would have understood this to teach that “truncat[ing]” the “output value” “to the selected subprecision” (Dockers, [0034]) is a third technique that can be applied to reduce precision in Dockser’s FPP, either alone (*i.e.*, truncating the output value by dropping mantissa bits after a full-precision multiplication) or in combination with either or both of register bit-dropping and/or logic bit-dropping (*i.e.*, unpowering bits less significant than the selected precision level in the output value that results from the register bit-dropping and/or logic bit-dropping, where each of those unpowered bits in the output value may or may not have already been forced to zero by the register and/or logic bit-dropping).

(1) Dockser’s Register Bit-Dropping Meets Limitation [1B2]

271. A POSA would have understood that Dockser’s “conventional floating-point multiplier” (Dockser, [0020]) multiplies two floating-point operands by (in relevant part) adding their exponents and multiplying their mantissas, as the ’273 patent acknowledges was the “traditional floating point method[.]” ’273 patent, 14:62-65 (“[t]o multiply values” by “*traditional floating point methods*[,] [t]he digital *exponents are summed* using a binary arithmetic adder, a standard digital technique. The analog *mantissas are multiplied*.” This is consistent with a POSA’s understanding of how floating-point multiplication, including IEEE-754 floating-point multiplication, is and was conventionally performed. *See, e.g.,* Tong, 274-275. This conventional technique applies basic properties of multiplication and exponents where $(M_1 \times 2^{E_1}) \times (M_2 \times 2^{E_2}) = (M_1 \times M_2) \times 2^{E_1+E_2}$. In that equation, M_1 and E_1 are the mantissa and exponent, respectively, of one floating-point number, and M_2 and E_2 are the mantissa and exponent, respectively, of another.

272. As I explained above in Section VI.B.4.c, Dockser reduces precision by dropping mantissa bits. *See, e.g.,* Dockser, [0002] (“The precision of the floating-point processor is defined by the number of bits used to represent the mantissa. The more bits in the mantissa, the greater the precision.”), [0003] (“[T]here is a need in the art for a floating-point processor in which the reduced

precision, or subprecision, of the floating-point format is selectable”), [0026] (“The subprecision select bits may be used to reduce the precision of the floating-point operation. This may be achieved in a variety of ways. In one embodiment, the floating-point controller 130 may cause power to be removed from the floating-point register elements for the excess bits of the fraction that are not required to meet the precision specified by the subprecision select bits.”). Because Dockser reduces precision by dropping mantissa bits, a POSA would have understood that the mantissa multiplication in Dockser produces error relative to an “exact mathematical calculation” of the product of the two operands with their full 23 fraction bits in the IEEE-754 single-format floating-point format, as I explain below in paragraphs 273-285.

273. Specifically, Dockser’s FPP receives operands into registers 115 as IEEE-754 single-format floating-point format numbers with 23 mantissa bits that represent the fractional part of the mantissa (as I discussed in Section VI.A above, the “1” in the mantissa’s integer part is implied), but Dockser’s register bit-dropping technique (which occurs at the registers, as shown in the annotated version of Figure 1 below) removes power from the register storage elements for the “excess” least-significant mantissa bits to the right of the selected precision level shown in Fig. 2 (reproduced in paragraph 276 below). In Dockser’s FPP, each of the “register locations” in Dockser’s “floating-point register file” is

“configured to store an operand for a floating-point operation.” Dockser, [0016]. Because “[e]ach register location 200 is configured to store a 32-bit binary floating-point number, in an IEEE-754 32-bit single format,” Dockser, [0017], a POSA would have understood that Dockser’s FPP receives operands into registers 115 as IEEE-754 single-format numbers with 23-bit mantissa fractions (the reference to a “24-bit fraction” in paragraph [0017] would have been understood as a typographical error, as I explain in paragraph 261 above).

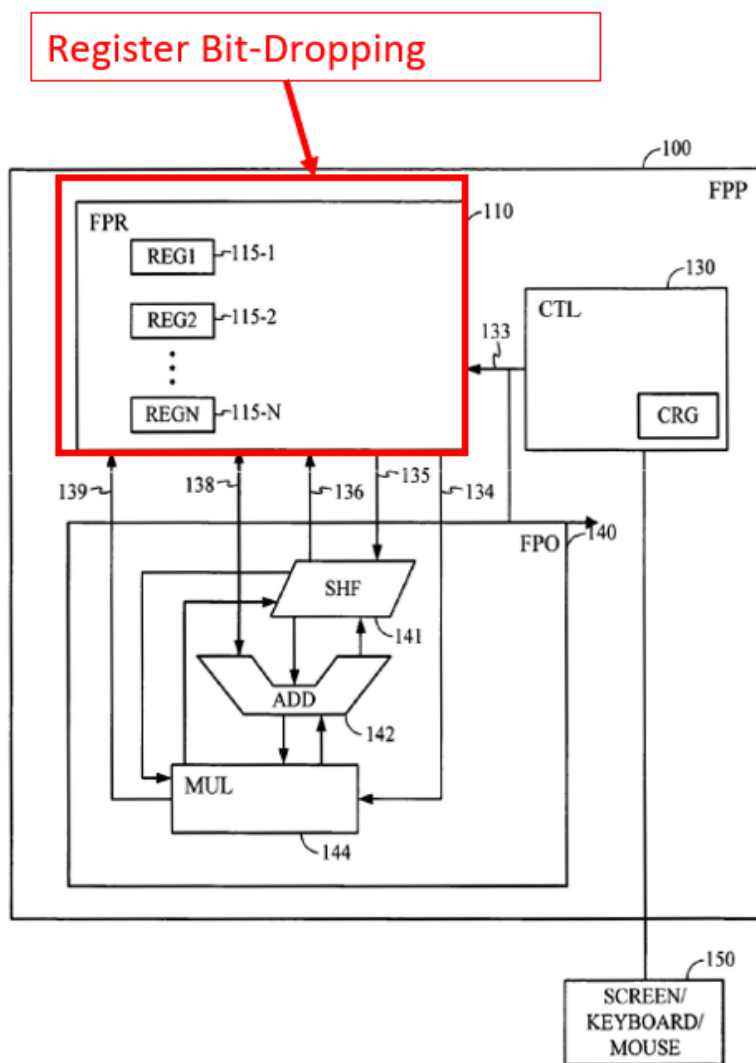


FIG. 1

274. Dockser’s register bit-dropping technique removes power from the storage elements in the register file that would store the “excess” mantissa fraction bits. Dockser, [0006], [0026]. In Dockser’s register bit-dropping technique, “the selection of the subprecision result[s] in one or more excess bits for each of said one or more of the floating-point numbers, the one or more excess bits being stored in one or more of the storage elements of the floating-point register,” and the floating-point controller is “configured to remove power from the storage elements for the one or more excess bits.” Dockser, [0006]; *see also* Dockser, [0026] (“the floating-point controller 130 may cause power to be removed from the floating-point register elements for the excess bits of the fraction that are not required to meet the precision specified by the subprecision select bits”).

275. To accomplish removal of power from register elements, Dockser explains that “[a] series of switches may be used to remove and apply power to the floating-point register elements.” Dockser, [0018]. These switches “may be internal or external to the floating-point register 110 and the floating-point operator 140” and “may be field effect transistors or any other type of switches.” Dockser, [0018].

276. A POSA would have understood that removing power from excess mantissa bits involved removing power from bits to the *right* of the selected precision level, as shown in Figure 2 (below, annotated), since Dockser retains

power in the “MSBs [most significant bits]” that are “required” to meet the selected subprecision (Dockser, [0026]), and “[i]n accordance with [the] standard convention, the floating-point register stores in order the bits that make up each number, ranging from a rightmost LSB to a leftmost MSB” (Dockser, [0028]).

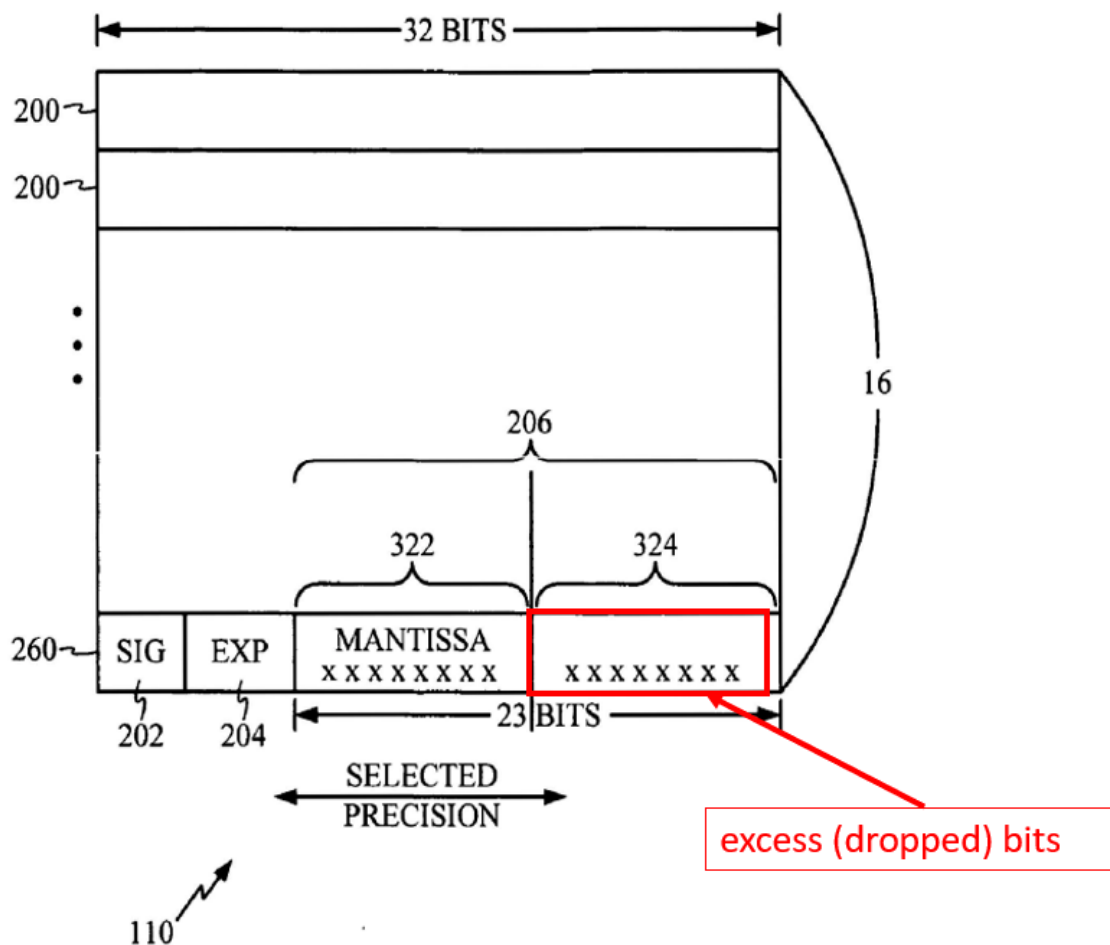


FIG. 2

277. “By way of example,” Dockser teaches that if a precision level retaining 9 mantissa fraction bits is selected, “[p]ower can be removed from the floating-point register elements for the remaining 14 [mantissa] fraction bits.”

Dockser, [0026]. Dockser refers to this example as an example where “the subprecision required for the floating-point operation is 10-bits,” but explains that in this example, “only the 9 commonly significant bits (MSBs) of the fraction are required; the hidden or integer bit makes the tenth.” Dockser, [0026].

278. A POSA would have understood that the output of the unpowered storage elements in Dockser’s FPP would be tied to zero voltage (*e.g.*, ground), making those 14 “excess” bits all zeroes. *See* Dockser, [0026] (“the floating-point controller 130 may cause power to be removed from the floating-point register elements for the excess bits of the fraction”), [0029] (referring to element 324 as “unpowered bits”), Fig. 2.

279. For evidence corroborating that it was a POSA’s background knowledge that unpowered elements should be tied to ground to prevent them from “floating” and drawing unacceptably large amounts of power and shorting the power supply, *see, e.g.*, Hawkins (Ex. 1020), 1:13-2:15 (*e.g.*, “Complimentary metal-oxide-semiconductor (CMOS) has become popular when used in low power integrated circuit devices” [like Dockser’s]. ... In order to retain the low power advantages of a CMOS device, it is important to ensure the gate terminal voltage not be allowed to float during periods of non-use. If the gate terminal voltage is left to float between, *e.g.*, a power supply and ground, then that device as well as possibly other connected devices may turn on thereby causing momentary shorting

of the power supply. ... The gate terminal of a CMOS transistor which receives a conductor having a floating value signal will draw unacceptably large amounts of power. CMOS devices draw the least amount of power when their inputs are at the upper supply level... or at ground voltage level. If the gate terminals of those devices are allowed to float between power and ground, then significantly larger amounts of power will be consumed.”); Youngs (Ex. 1060), [0005] (“Modern processors” [like Dockser’s FPP] “are typically fabricated using Complementary Metal Oxide Semiconductor (CMOS) circuitry.”); Flynn (Ex. 1019), [0003] (“integrated circuit design” tries “to prevent floating inputs or outputs arising when portions of the integrated circuit are powered down which could otherwise result in unpredictable or incorrect operation elsewhere.”)

280. For evidence corroborating that it was a POSA’s background understanding that tying the outputs of the unpowered storage elements for “excess” bits to zero voltage would make those bits zeroes, *see, e.g.*, Cohen (Ex. 1016), 3:63-66 (“A binary ‘1’ is typically represented by a high voltage, such as 5 or 3.3 volts, while ***a binary ‘0’ is typically represented by a low voltage, such as 0 volts or ground.***”), 4:51-55 (“A ‘1’ is a high logic value, typically represented ***in a digital computer system*** as signal with a high voltage, such as 5 or 3.3 volts, while ***a ‘0’ is a logic low value, typically represented by a low voltage such as 0 volts or ground.***”). Dockser’s Figures also use circles to represent unpowered bits, which

visually indicates to the POSA that those bits are zeroed. Dockser, FIGs. 3A-3B, [0029].

281. To the extent Dockser is not considered to disclose that the outputs of the unpowered register storage elements would be tied to ground to represent “0,” that would have been the straightforward, well-known, conventional, and obvious way to implement Dockser’s described unpowering of register storage elements. This is corroborated, *e.g.*, by the teachings of Hawkins, Youngs, Flynn, and Cohen that I discussed above. A POSA would have understood that there were only three options with regard to the outputs of un-powered storage elements: tie them to low voltage (ground), tie them to high voltage (supply), or allow them to “float” in between. Of these three options, a POSA would have understood the third (“floating”) to be unacceptable for the reasons I explained in paragraphs 279-280 above, and a POSA would have understood the first option (ground) to be far more typical, conventional, and commonplace for un-powered circuit elements than the second option (supply). A POSA would have understood that choosing to tie the outputs to ground is consistent with how Dockser discloses treating other unnecessary bit elements (*e.g.*, Dockser, [0029]: “[t]he carry-out C from the last powered down stage 310_i is forced to zero”) and would have had the predicted and desirable result of allowing the FPP to perform floating-point operations at reduced precision while saving power.

282. As representative examples of Dockser’s register bit-dropping technique dropping least-significant mantissa fraction bits to zero, the 23-bit sequence 01100000111100111001110 (representing a mantissa of 1.3787171840667724609375) would become 01100000100000000000000 (representing a mantissa of 1.376953125) by dropping the 14 rightmost bits to zero, and another 23-bit sequence 01000100110111000001000 (representing a mantissa of 1.26898288726806640625) would become 010001001000000000000000 (representing a mantissa of 1.267578125) by dropping the 14 rightmost bits to zero.

283. When Dockser’s register bit-dropping technique is used for multiplication, a pair of operands with “excess” bits dropped from the registers are multiplied together, producing a reduced-precision output that differs from the exact product of the original 32-bit operands. *See* Dockser, [0024] (“Upon ***receiving the operands from the floating-point register file*** 110, one or more computational units in the floating-point operator 140 may ***execute*** the instructions of the ***requested floating-point operation on the received operands***, at the subprecision selected by the floating-point controller 130.”), [0025] (“a software selectable mode may be used to reduce the precision of the floating-point operations under program control or as explained above, the instructions provided to the floating-point processor 100 may include a programmable control field

containing the subprecision select bits. *The subprecision select bits are written to the control register 137, which in turn controls the bit length of the mantissa for each operand during the floating-point operation.* Alternatively, the subprecision select bits may be written to the control register 137 directly from any suitable user interface, including for example but not limited to a monitor screen/keyboard or mouse 150 shown in FIG. 1. In another embodiment of the floating-point processor 100, the subprecision selection bits may be written to the control register 137 directly from the main processor, or its operating system.”), [0026] (“the floating-point controller 130 may cause *power to be removed from the floating-point register elements for the excess* bits of the fraction that are not required to meet the precision specified by the subprecision select bits”).

284. For instance, if the two example 23-bit-fraction mantissas from paragraph 282 above were multiplied, the exact mantissa product would be 1.7495685129631510790204629302025; whereas the reduced-precision mantissa product output from Dockser’s register bit-dropping technique would be 1.745395660400390625 (resulting from multiplying the mantissas represented by the fraction bit sequences 011000001000000000000000 and 010001001000000000000000).

285. As can be seen from the example above, when bits in the mantissas of the operands are dropped to zero, Dockser’s output mantissa differs from the exact

mantissa product. Because of this, the numerical value represented by the output floating-point number (which is composed of the output mantissa, exponent, and sign) differs from the exact product of the original floating-point operands, because the exact product of the original operands has a mantissa equal to the product of the original operand mantissas before bits were dropped to zero.

286. A POSA would have understood, by straightforward math that I detail in Appendix I.A below, that the relative error amount (the “Y” percentage recited in limitation [1B2]) of any floating-point number output from Dockser’s reduced-precision multiplication is the same as the relative error amount of its mantissa, and is independent of its exponent and sign.

287. A POSA would thus have understood that Dockser’s register bit-dropping technique produces error, the amount of which depends on the number of mantissa bits dropped—the more bits dropped, the greater the error (except for the small set of inputs in which the bits that are dropped were 0s to begin with).

288. Claim 1 of the ’273 patent characterizes the claimed execution unit using *performance* characteristics of a minimum relative error amount (Y) and a minimum fraction (X) of the possible valid inputs that meet Y. Claim 1 recites no *structural* characteristic of the execution unit that a POSA could evaluate to determine whether Dockser’s execution unit (or *any* execution unit) meets the claim’s limitations relating to the execution unit’s level of (im)precision. Based on

my review of the specification, the specification provides no guidance on how to evaluate any execution unit's structure to determine whether it meets the performance characteristics.

289. A POSA asked whether Dockser's FPP meets the claimed imprecision performance characteristics recited in limitation [1B2] would have understood that Dockser suggests that any desired number of mantissa bits can be dropped (*see* my discussion in Section VIII.G below), and would have understood that numerous obvious implementations of Dockser's FPP drop sufficient bits to yield an execution unit that is imprecise enough to meet the claimed minimum imprecision performance characteristics X and Y. If asked to quantify how many bits would need to be dropped to meet the claimed X and Y, a POSA would have taken one of two approaches, as I explain in Sections VI.B.4.c(1)(i)-VI.B.4.c(1)(ii) below.

(i) Software Demonstration That Dockser's Register Bit-Dropping Meets Limitation [1B2]

290. Given the massive number of possible inputs to Dockser's FPP (there are over 70 trillion possible pairs of normal IEEE-754 single-format mantissas, even without considering combinations with possible exponents and signs), a POSA would have performed Dockser's FPP operation in software to determine the fraction X of all possible valid inputs that produce at least the relative error Y recited in limitation [1B2], when a given number of mantissa bits are dropped.

291. As I detail in Appendix I.B below, I wrote such a software program (which a POSA would have understood how to write), that performs floating-point multiplication operations the way Dockser's register bit-dropping technique does when dropping the 14 least-significant mantissa bits to zero as Dockser teaches. Dockser, [0026].

292. My program tested all possible valid pairs of normal IEEE-754 single-format operands (by testing all possible valid mantissa pairs, because the relative error Y is independent of exponent and sign, as I explain in Appendix I.A below), and demonstrates that Dockser's register bit-dropping technique, when performed with a selected precision level retaining 9 mantissa fraction bits as Dockser ([0026]) discloses, produces at least $Y=0.05\%$ relative error for 92.1% of possible valid inputs (greater than $X=5\%$), meeting limitation [1B2].

**(ii) Pencil-and-Paper Algebraic Demonstration
That Dockser's Register Bit-Dropping Meets
Limitation [1B2]**

293. A POSA would also have understood algebraically that Dockser's register bit-dropping technique meets limitation [1B2], by examining the absolute *minimum* relative error produced by dropping to zero the mantissa bits at certain bit positions. A POSA would have examined absolute minimum relative error since the '273 patent's claims are met so long as the recited minimum relative error

(“at least $Y=0.05\%$ ”) is equaled *or exceeded* for some minimum percentage of the possible valid inputs (“at least $X=5\%$ ”).

294. As I detail mathematically in Appendix I.C below, over 12% of all possible valid normal IEEE-754 single-format operands have a zero as their most-significant (left-most) mantissa fraction bit and ones as their tenth and eleventh fraction bits. When Dockser’s register bit-dropping technique is performed while retaining only nine fraction bits (thus dropping the tenth and eleventh bits as well as the remaining 12 less-significant mantissa bits), *every* input in that 12% produces at *minimum* 0.097% relative error, which meets limitation [1B2]’s recited X and Y values.

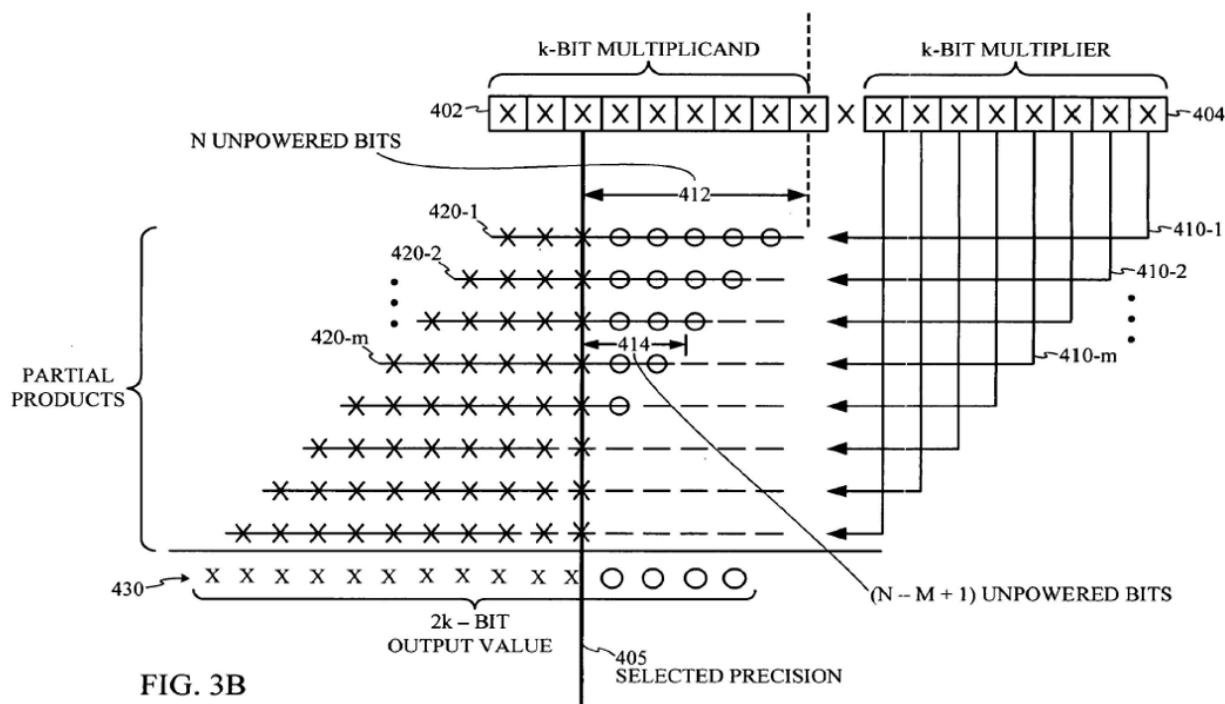
(2) Dockser’s Multiplier Logic Bit-Dropping Meets Limitation [1B2]

295. Dockser’s second precision-reduction technique removes power from bits within the FPO’s logic that multiplies the mantissas of the operands. *See* Dockser, [0007] (“...the floating-point controller is further configured to remove power a portion of the logic that would otherwise be used to process the one or more excess bits”), [0027] (“...the logic in the floating-point operator 140 corresponding to the excess mantissa bits do not require power. Thus, power savings may be achieved by removing power to the logic in the floating-point operator 140 that remains unused as a result of the subprecision selected”).

Dockser describes how the multiplier operates at reduced precision in paragraphs [0030]-[0034], which I discuss below.

296. Dockser’s floating-point multiplication logic operates conventionally, computing the mantissa product by generating and adding together “partial products” in a manner similar to pencil-and-paper long multiplication. See Dockser, [0030] (“FIG. 3B is a conceptual diagram illustrating an example of a floating-point multiplication operation with power being selectively applied to logic in the floating-point operator. The floating-point multiplication operation is performed in the floating-point multiplier MUL, shown in FIG. 1 with reference numeral 144....Binary multiplication as illustrated in FIG. 3B is basically a series of additions of shifted floating-point numbers. In the illustrated embodiment, binary multiplication is performed between a k-bit multiplicand 402 and a k-bit multiplier 404, using a shift-and-add technique.”), [0031] (“As in the case of floating-point addition, floating-point multiplication is performed in a series of stages, illustrated in FIG. 3B as 410-1, . . . , 410-m....***one partial product is generated for every bit in the multiplier 404***, a partial product 420-i being generated during a corresponding stage 410-i. If the value of the multiplier is 0, its corresponding partial product consists only of 0s; if the value of the bit is 1, its corresponding partial product is a copy of the multiplicand. ***Each partial product 420-i is left-shifted***, as a function of the multiplier bit with which it is associated,

after which the operation moves on to the next stage. *Each partial product can thus be viewed as a shifted number.*”), FIG. 3B.



297. I note (and a POSA would have understood) that in paragraphs [0030]-[0031], the reference to “multiplier 404” (“k-bit multiplier 404” in FIG. 3B) does not refer to the “multiplier” hardware (which is designated with a different reference number, 144, as shown in Figure 1). Instead, a POSA would have understood that in that context, the “multiplier 404” refers to one of the two numbers being multiplied by the multiplier hardware, with the other number being referred to as the “multiplicand” and labeled with element number 402, as shown in the annotated version of Figure 3B above. *See Dockser*, [0030] (“In the

illustrated embodiment, binary multiplication is performed between a k-bit multiplicand 402 and a k-bit multiplier 404...). A POSA would have understood that in the field of mathematics, the two numbers that are multiplied in a multiplication operation were sometimes referred to as the “multiplier” and “multiplicand.”

298. I provide the simple example below using example 5-bit mantissa fractions to illustrate the conventional process that Dockser's multiplier uses to compute a product of mantissas. Each "partial product" results from multiplying the multiplicand (Mantissa A in my example below) by the "0" or "1" at each position in the multiplier (Mantissa B in my example below), and the partial products are summed to compute the output mantissa product. The binary point (binary equivalent of a decimal point) in the multiplicand, multiplier, and product is near the left-hand side of my illustration below because the numbers being multiplied are mostly composed of fractional bits.

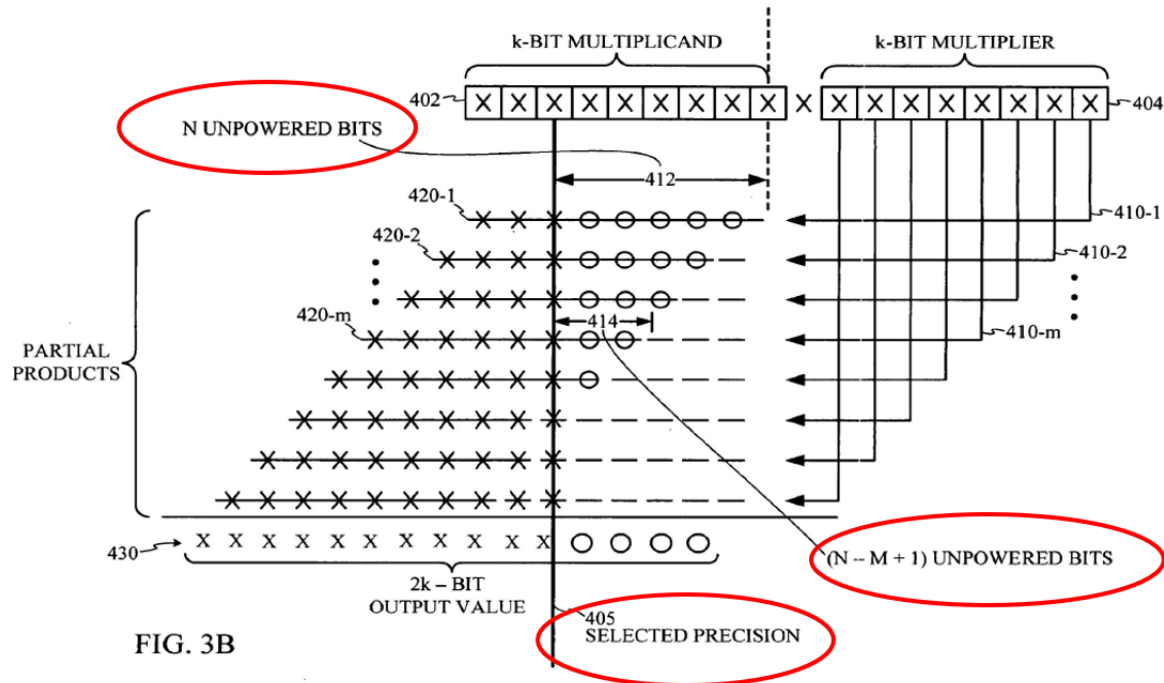
	1	.	0	1	1	0	0					Mantissa A = 1.375	
x	1	.	0	1	0	0	1					Mantissa B = 1.28125	
								1	0	1	1	0	0
								0	0	0	0	0	0
				0	0	0	0	0	0	0	0		
			1	0	1	1	0	0					
		0	0	0	0	0	0						
(+)	1	0	1	1	0	0							
	1	.	1	1	0	0	0	0	1	1	0	0	Mantissa product = 1.76171875

299. In Dockser's logic bit-dropping technique, power is removed from the multiplier logic that implements the partial product bits to the right of line 405 (shown in FIG. 3B reproduced below), which corresponds to the precision level selected for the output value 430. *See* Dockser, [0032]-[0034]. Dockser explains that "[i]n the embodiment illustrated in FIG. 3B, the selection of a desired reduced precision by the controller 130 is indicated with a line 405"—*i.e.*, line 405 indicates the selected precision level. Dockser, [0032]. The multiplication produces an "output value 430," and Dockser teaches that the selected precision level (indicated by line 405) corresponds to the number of bits to be nonzero in the output value 430: "The output value 430 may be truncated to the selected subprecision, *i.e.* any of the bits of the output value 430 that are in less the selected precision may be truncated, to generate a truncated output number characterized by the selected precision." Dockser, [0034]. This is as further shown in Figure 3B (reproduced below), where the "selected precision" line 405 is drawn between the bits in the output value that will be zeroes and the bits in the output value that may be nonzero. Thus, for example, when the selected precision level is 9 retained mantissa fraction bits (Dockser, [0026]), the line 405 is placed between the 9th mantissa fraction bit in the output value 430 and the 10th mantissa fraction bit in the output value 430 (which is to be zeroed).

300. Dockser's FIG. 3B (reproduced below) also illustrates how the line 405 corresponding to the precision level selected for the output value 430 extends to separate the powered bits from the unpowered bits in the partial products.

Dockser, [0032]. The powered partial product bits to the left of line 405 are those partial product bits corresponding to the bit positions that are above (to the left of) the selected precision level in the output value 430, and the unpowered partial product bits to the right of line 405 are those partial product bits corresponding to the bit positions that are below (to the right of) the selected precision level in the output value 430: "[P]ower may be removed from the logic used to implement the stages to the right of the line 405. Power is only applied to the stages that are actually needed to support the selected subprecision, i.e., the stages to the left of the line 405." Dockser, [0032]. Thus, in each partial product, a successively smaller number of bits, starting with some number N , is turned off, because as each product is shifted farther and farther to the left, more of the bits of that product will fall to the left of line 405, meaning that they remain powered. *See* Dockser, [0033] ("As seen from FIG. 3B, for the first partial product 420-1, the logic for a number of bits N , shown using reference numeral 402, is unpowered. For the second partial product, the logic for $N-1$ bits is unpowered, and so forth. For the m -th partial product or shifted floating-point number 420- m , the logic for a number $(N-m+1)$ of bits, shown using reference numeral 414, is unpowered."). A POSA would have

understood that the number N can be larger than the number of bits k in the multiplicand and/or multiplier, *e.g.*, if a selected precision level 405 is chosen that is smaller than k , such that the output value 430 will have fewer than k nonzero bits and more than k zeroed bits.



301. As with the bits from which power is removed in the registers in Dockser's register bit-dropping technique, a POSA would have understood Dockser to disclose that the bits from which power is removed in the multiplier logic are dropped to 0s, or alternatively that this would have been the conventional and obvious way to implement the power removal that Dockser describes, for the same reasons I discussed in Section VI.B.4.c(1) above (*see* paragraphs 278-281

above, explaining that a POSA would have understood that unpowered bits would be tied to ground and represent 0s).

302. I illustrate an example of Dockser's multiplier logic bit-dropping technique below using the same exemplary operand mantissas that I used in my example in Section VI.B.4.c(1) above. In the example below, dropping bits in the partial products to the right of the selected precision (Dockser's line 405, illustrated below as a red dashed line at the example selected precision level of 9 mantissa fraction bits retained in the output value) produces an output mantissa product of 1.744140625, which differs from the exact product of 1.7495685129631510790204629302025 that would have resulted from multiplying the example operand mantissas without reducing precision.

	1	.	0	1	1	0	0	0	0	0	1	1	} Operand mantissas				
X	1	.	0	1	0	0	0	1	0	0	1	1					
<hr/>																		
													0	0	0	0	} Partial products with unpowered bits
											1		0	0	0	0	
								0	0				0	0	0	0	
								0	0	0			0	0	0	0	
								1	0	1	1		0	0	0	0	
								0	0	0	0	0	0	0	0	0	
								0	0	0	0	0	0	0	0	0	
								0	0	0	0	0	0	0	0	0	
								1	0	1	1	0	0	0	0	0	} Mantissa product = 1.74414....
								0	0	0	0	0	0	0	0	0	
(+)	1	.	0	1	1	0	0	0	0	0	1	0	0	0	0	0	
	1	.	1	0	1	1	1	1	1	0	1	0	0	0	0	0	
<hr/>																		
	1	.	1	0	1	1	1	1	1	0	1	0	0	0	0	0	

Selected precision = 9 fraction bits

Mantissa product = 1.74414....

303. A POSA would have understood that the relative error introduced by Dockser's logic bit-dropping is dependent on the number of mantissa bits dropped as specified by the selected precision level 405—the more bits dropped, the greater the error. This is because, as can be seen in the example above, as more bits are dropped, more bits in the partial products become 0, which makes the overall product of the mantissas smaller. This in turn makes the relative error compared to the exact mathematical calculation of the mantissa product bigger, since the relative error is the percentage *difference* between the exact product and the product produced by Dockser's mantissa multiplier, and that percentage difference gets bigger as the two numbers get farther away from each other in value. This in turn makes the relative error between the exact mathematical calculation of the floating-point product and the product produced by Dockser's FPP bigger, since the relative error in the floating-point product is the same as the relative error in the mantissa product, as I explain in Appendix I.A below.

304. If asked whether Dockser's FPP using a selected precision level retaining 9 bits (as Dockser [0026] teaches) in the output mantissa by dropping less-significant bits in the multiplier logic meets the X and Y recited in limitation [1B2], given the massive number of possible inputs that Dockser's FPP supports (as I discussed in Section VI.B.4.c(1)(i) above), a POSA would have written a software program.

305. As I describe in Appendix I.D below, I wrote such a software program (which a POSA would have understood how to write) that performs floating-point multiplication operations in the way Dockser’s logic bit-dropping technique does by dropping to zero the mantissa partial product bits that are less significant than the 9-bit selected precision level of the output value. My program tested all possible valid pairs of normal IEEE-754 single-format operands (by testing all possible valid mantissa pairs, which is sufficient to test all possible operand pairs, as I explain in Section VI.B.4.c(1)(i) above and Appendix I.D below), and the program demonstrates that Dockser’s multiplier logic bit-dropping technique, when performed with a selected precision level that retains 9 nonzero mantissa fraction bits in the output, produces at least $Y=0.05\%$ relative error for 99.35% of possible valid inputs (which is greater than $X=5\%$), meeting limitation [1B2].

306. In the alternative mapping where Dockser’s FPO meets the claimed “execution unit” and signal 134 meets the claimed “first input signal” (*see* Section VI.B.1 above, where I discuss the alternative mapping where the FPO inside the FPP meets the “execution unit” in limitation [1A1], and Section VI.B.2 above, where I explain the corresponding alternative mapping of signal 134 in Dockser’s Figure 1 to the “first input signal” in limitation [1A2])), the operand numbers that are input via 134 (the claimed “first input signal” to the FPO) are the same as those input to the FPP registers when Dockser’s logic bit-dropping technique is used

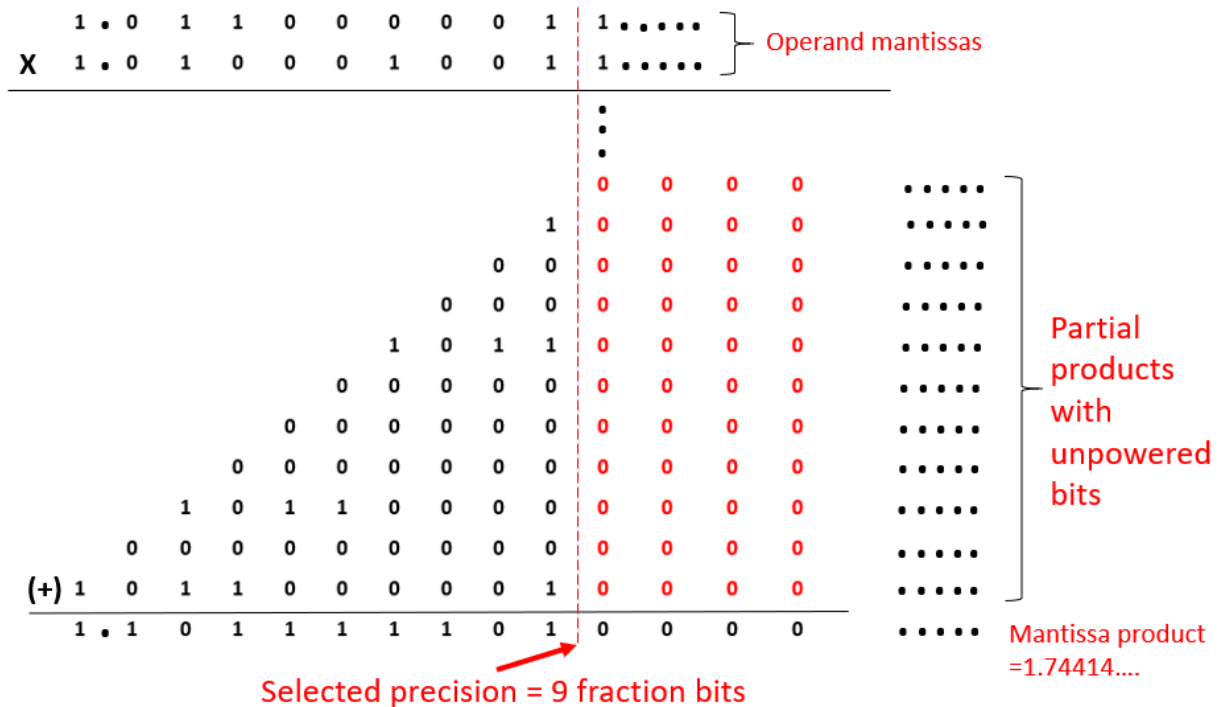
alone. This is because when Dockser's logic bit-dropping technique is used alone, no register bit-dropping is performed, and operands are passed unchanged from the registers to the multiplier logic. Furthermore, in this mapping, the same output numbers from the FPO are output from the FPP. This is because, as I explain in paragraphs 236-237 above, the multiplication result that the FPP sends back to memory from the register file is the output number from the FPO, which gets stored in the register file. Therefore, my program demonstrates that Dockser meets [1B2] under either mapping.

(3) Performing Register and Multiplier Logic Bit-Dropping Together Also Meets [1B2]

307. In an implementation of Dockser's FPP that drops bits in *both* the registers and the multiplier logic, where Dockser's FPP is the claimed "execution unit," the amount of relative error is the same as produced by the logic bit-dropping alone, if the same selected precision level (retaining the same number of mantissa fraction bits) is applied at the registers and in the logic. This is because at least the same bits that are dropped to zero in the register bit-dropping are dropped to zero in the corresponding partial products in the logic bit-dropping, for the same selected precision level, as a POSA would have understood.

308. For example, if the selected precision level is 9 mantissa fraction bits, as in Dockser's [0026], then register bit-dropping causes all but the 10 highest-order partial products (all but the 10 partial products that are lowest toward the

bottom of the illustration of partial products in, *e.g.*, FIG. 3B and my illustration below) to be entirely zero, because all but the 10 most-significant bits of the multiplier mantissa are dropped to zero. *See* Dockser, [0031] (“[O]ne partial product is generated for every bit in the multiplier 404, a partial product 420-i being generated during a corresponding stage 410-i. If the value of the multiplier is 0, its corresponding partial product consists only of 0s; if the value of the bit is 1, its corresponding partial product is a copy of the multiplicand”). Similarly, logic bit-dropping with the selected precision level at 9 mantissa fraction bits causes all but the 10 highest-order partial products to be entirely zero, as my example below illustrates.



309. Register bit-dropping with a selected precision level of 9 mantissa fraction bits also causes all but the 10 most-significant bits in every partial product to be zero, because all but the 10 most-significant bits of the multiplicand are dropped to zero. Similarly, logic bit-dropping with the selected precision level at 9 mantissa fraction bits causes all but the 10 most-significant bits in every partial product to be zero, as my example above illustrates. Thus a POSA would have understood that for the same selected precision level (*e.g.*, selected precision level of 9 mantissa fraction bits), all bits that are dropped to zero in the partial products because of the register bit-dropping are also dropped to zero in those partial products because of the logic bit-dropping. And the logic bit-dropping also drops some additional partial product bits to zero that are not dropped to zero because of the register bit-dropping—*e.g.*, the 10th most-significant bit (9th most-significant fractional bit) in the second highest-order partial product.

310. Therefore, because at least the same bits that are dropped to zero in the register bit-dropping are dropped to zero in the corresponding partial products in the logic bit-dropping, for the same selected precision level, a POSA would have understood that the amount of relative error from performing both register and logic bit-dropping together is the same as produced by the logic bit-dropping alone. Thus, a POSA would have understood that the results of the same software program that demonstrate that Dockser's multiplier logic bit-dropping alone meets

limitation [1B2] also demonstrate that Dockser’s register and logic bit-dropping, performed together at a selected precision level retaining 9 mantissa fraction bits as Dockser ([0026]) discloses, meet limitation [1B2].

C. Claim 2: “The device of claim 1, wherein the at least one first LPHDR execution unit comprises at least part of an FPGA”

311. Dockser’s execution unit comprises at least “part of...[an] *FPGA*,” meeting claim 2. Dockser, [0035] (“The various illustrative logical units, blocks, modules, circuits, elements, and/or components described in connection with the embodiments disclosed herein may be implemented or performed in a *floating-point processor that is part of ... a field programmable gate array (FPGA)* or other programmable logic component, discrete gate or transistor logic, discrete hardware components, or any combination thereof designed to perform the functions described herein.”)

D. Claim 21: “The device of claim 1, wherein the device includes memory locally accessible to the at least one first LPHDR execution unit.”

312. The ’273 patent describes a PE unit as “pair[ing] memory with arithmetic,” as having “memory *local* to each arithmetic unit,” and as implementing the memory via registers. ’273 patent, 16:31-56 (“...we choose a very general embodiment of an LPHDR machine. Our model of the machine is that it provides at least the following capabilities provides a small amount of

memory local to each arithmetic unit ... While we are thus showing that LPHDR arithmetic is useful for a broad range of designs, of which SIMD is only an instance, for purpose of discussion below, we call *each unit, which pairs memory with arithmetic, a Processing Element or ‘PE’*”), 10:34-57 (“FIG. 4 shows an example design for a PE 400 ... The PE 400 stores local data. The amount of *memory* for the local data varies significantly from design to design. ... Sometimes rarely changing values (Constants) take less room than frequently changing values (*Registers*), and a design may provide more Constants than Registers. ... Sometimes the situation is reversed ... Typical storage capacities might be tens or hundreds of arithmetic values stored in the *Registers* and Constants in each PE ... All of these variations may be reflected in embodiments of the present invention.”), 11:17-28 (“The operation of the PEs is controlled by control signals 412 a-d ... the control signals 412 a-d specify which Register or Constant memory values in the PE 400 or one of its neighbors to send to the data paths, which operations should be performed by the Logic 406 or Arithmetic 408 or other processing mechanisms, where the results should be stored in the Registers ... and so on. These matters are well described in the literature on SIMD processors.”), FIG. 4 (reproduced below, annotated).

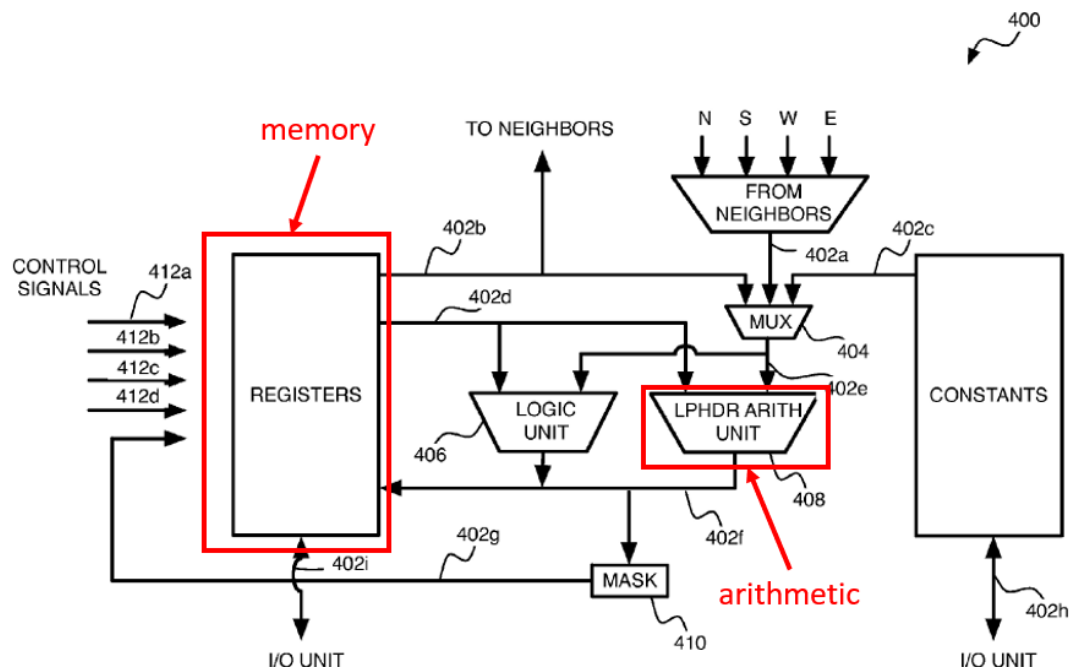


FIG. 4

313. Dockser's FPP similarly has local registers 110. Dockser, [0015] ("The floating-point processor 100 includes a *floating-point register file (FPR) 110* ..."), [0016] ("The floating-point register file 110 may be any suitable storage medium. In the embodiment shown in FIG. 1, the floating-point register file 110 *includes several addressable register locations* 115-1 (REG1), 115-2 (REG2), . . . 115-N (REGN), each configured to store an operand for a floating-point operation."). *See also* Section VI.B.1 above (where I explain that Dockser's FPP, or alternatively the FPO in Dockser's FPP, is an LPHDR execution unit).

314. Therefore, Dockser’s device “includes memory locally accessible to” Dockser’s execution unit, as claim 21 of the ’273 patent recites. Dockser’s FPP registers are locally accessible to Dockser’s FPP and its FPO.

E. Claim 22: “The device of claim 1, wherein the device is implemented on a silicon chip”

315. Dockser’s FPP can be “part of... an application specific integrated circuit (ASIC)” that meets the “device” of claim 1 of the ’273 patent. *See* Dockser, [0035] (“The various illustrative logical units, blocks, modules, circuits, elements, and/or components described in connection with the embodiments disclosed herein may be implemented or performed in a floating-point processor that is part of a general purpose processor, a digital signal processor (DSP), *an application specific integrated circuit (ASIC)*, ... or any combination thereof designed to perform the functions described herein.”); *see* Section VI.B.1 above (where I explained that Dockser’s ASIC meets the claimed “device”).

316. Implementing Dockser’s integrated circuit on a silicon chip would have been the typical, conventional, and obvious implementation; thus, Dockser’s “device is implemented on a silicon chip” as claim 22 of the ’273 patent recites. *See* Steffen (Ex. 1054), [0005] (“An integrated circuit conventionally comes in the form of a silicon chip[.]”); Kacker (Ex. 1055), [0005] (“Integrated circuits (chips) are generally made of silicon on which electronic circuits are fabricated.”).

F. Claim 23: “The device of claim 1, wherein the device is implemented on a silicon chip using digital technology.”

317. Dockser’s integrated-circuit device is “implemented on a silicon chip” as claimed. *See* my explanation in Section VI.E above. Dockser’s FPP (containing the FPO) is part of this integrated-circuit device and “process[es]... bits.” *See* Dockser, Abstract (“A method and apparatus for performing a floating-point operation with a floating-point processor having a given precision is disclosed. A subprecision for the floating-point operation on one or more floating-point numbers is selected. The selection of the subprecision results in one or more excess bits for each of the one or more floating-point numbers. Power may be removed from one or more components in the floating-point processor that *would otherwise be used to store or process* the one or more excess *bits*.”); *see* Sections VI.B.1, VI.E above (explaining that the FPP can be implemented as part of the ASIC). Therefore, Dockser’s integrated-circuit device “is implemented... using digital technology” as claim 23 of the ’273 patent recites.

G. Claim 24: “The device of claim 1, wherein the device further comprises a digital processor adapted to control the operation of the at least one first LPHDR execution unit.”

318. As I discussed in Section VI.B.1 above, Dockser’s computing system is one of the elements in Dockser that meets the “device of claim 1” recited in the ’273 patent’s claims. Dockser’s computing system includes a “main processor.” *See* Dockser, [0015], [0035]. Dockser states that “[t]he floating-point processor

100 may be implemented as *part of the main processor*, a coprocessor, *or* a separate entity *connected to the main processor* through a bus or other channel.”

Dockser, [0015]. Dockser also states:

The various illustrative logical units, blocks, modules, circuits, elements, and/or components described in connection with the embodiments disclosed herein may be implemented or performed in a *floating-point processor that is part of a general purpose processor*, a digital signal processor (DSP), an application specific integrated circuit (ASIC), a field programmable gate array (FPGA) or other programmable logic component, discrete gate or transistor logic, discrete hardware components, *or any combination thereof* designed to perform the functions described herein. *A general-purpose processor may be a microprocessor*, but in the alternative, the processor may be *any conventional processor, controller, microcontroller, or state machine. The processor may also be implemented as a combination of computing components, e.g., a combination of a DSP and a microprocessor*, a plurality of microprocessors, one or more microprocessors in conjunction with a DSP core, or any other such configuration.

Dockser, [0035]. Thus, a POSA would have understood that the “main processor” Dockser refers to in [0015] is a “general-purpose processor” referred to in [0035], and would also have understood that Dockser discloses implementing the FPP as part of or separate from and connected to the “main processor”/“general-purpose processor.” Dockser, [0035].

319. Dockser’s main processor writes “subprecision select *bits*” to the FPP’s “control register” (Dockser, [0025]), and thus a POSA would have understood the main processor a “digital processor” as recited in claim 24 of the ’273 patent—*e.g.*, because Dockser’s main processor communicates via “bits.” Alternatively, to the extent Dockser is not considered to expressly disclose that its main processor is “digital,” that would have been the typical and obvious implementation of Dockser’s “conventional processor.” Dockser, [0035]. For corroborating evidence, *see* Nelson (Ex. 1023), 1:16-48 (explaining the “main processor” of “[m]any portable computers” like Dockser’s is typically “of the type commercially available from Intel and other suppliers called generically the ‘486’ type” or “some others such as the Intel PENTIUM™ processors,” each of which a POSA would have understood is a well-known “digital processor” as claim 24 of the ’273 patent recites).

320. By writing “subprecision select bits” to the FPP’s “*control* register,” Dockser’s main processor is adapted to control the operation of the FPP (and its FPO) by specifying its precision level. Dockser, [0018], [0025]. Dockser explains that “the floating-point controller 130 may be used to select the subprecision of the floating-point operations using a control signal 133.” Dockser, [0018]. A “control register (CRG) 137” in the floating-point controller “may be loaded with *subprecision select bits* for example *transmitted in* the control field of one or more

instructions.” Dockser, [0018], Fig. 1; *see also* Dockser, [0025] (“The control register [CRG] 137, which is shown in the floating-point controller 130....”). “[T]he subprecision selection bits may be written to the control register 137 directly from the *main processor.*” Dockser, [0025].

H. Claim 26: “The device of claim 1, wherein the device is part of a mobile device.”

321. Dockser’s computer system and ASIC (both of which meet the ’273 patent’s “device of claim 1” as I discussed in Section VI.B.1 above) are “part of a mobile device” as claim 26 recites—*e.g.*, a “wireless telephone[],... (PDA),... pager[],...[etc.].” Dockser, [0003].

I. Claim 28: “The device of claim 1, wherein the at least one first LPHDR execution unit represents numbers using a floating point representation.”

322. As I discussed in Section VI.B.2 above, Dockser’s execution unit “represents numbers using a floating point representation,” as claim 28 of the ’273 patent recites. *See* Section VI.B.2 (Dockser’s execution unit is the “floating-point processor,” or alternatively the “floating-point operator,” that operates on and produces floating-point numbers).

VII. CLAIMS 1-2, 21-24, 26, 28, AND 32-33 WOULD HAVE BEEN OBVIOUS OVER DOCKSER AND TONG

A. Tong (Ex. 1008)

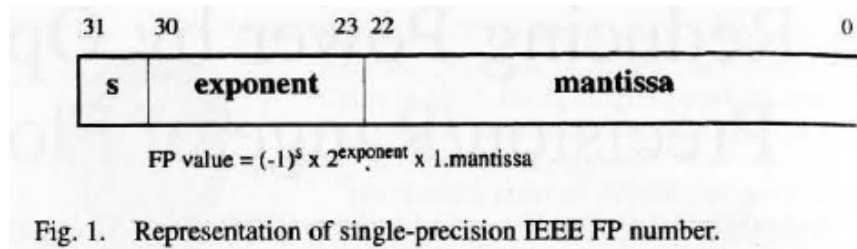
323. Tong’s face bears a copyright date of 2000 and indicates that it was published by the Institute for Electrical and Electronics Engineers (IEEE) in *IEEE*

Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 8, No. 3, June 2000. I am a Senior Member of the IEEE and am personally familiar with the IEEE. The IEEE is a well-known, reputable, and well-respected organization that publishes respected technical papers in various journals in fields including computer science, computer engineering, and electrical engineering. Based on my knowledge as a POSA and my familiarity with and membership in the IEEE, I am aware that POSAs interested in the subjects addressed by Tong and the '273 patent would have known how to search for relevant papers in IEEE publications and would have been led to access Tong, either by obtaining a physical copy of the relevant issue of *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* or by downloading it from the IEEE website.

324. Tong, like Dockser (*see* Section VI.B.4.c above and Section VIII.G below), confirms that the number of mantissa bits used in a high-dynamic-range floating-point execution unit was a well-known result-effective variable impacting both power consumption and precision. Tong, 278 (“[P]ower dissipation in an FP [floating-point] unit can be reduced by using fewer bits in the FP representation. However, fewer bits reduces precision”).

325. A POSA would have understood that Tong uses “mantissa” as shorthand for what Dockser calls the “fraction” portion of the mantissa of an IEEE-754 single-format floating-point number—*i.e.*, the portion of the mantissa to the

right of the binary/radix point. *See* Tong, 274 (“IEEE-754 specifies that any single-precision FP number be represented using 1 sign bit, 8 bits of exponents, and 23 bits of mantissa... An IEEE single-precision number is represented in Fig. 1.”), Figure 1.



326. Focusing on “signal processing applications” which have “long been known” to be able to “get by with less precision/range than full FP” (Tong, 273), Tong explains that “[s]ince multiplication is one of the most frequent operations in signal processing applications” (Tong, 274) and “the mantissa multiplier always consumes more power than all other blocks” in an FP multiplier (Tong, 275-276), “reduction in the mantissa bitwidth is the most effective means of reducing power dissipation” (Tong, 277). Tong explains that “Floating-point (FP) hardware provides a wide dynamic range of representable real numbers,” and notes that “FP hardware’s performance, simplified programming model, and adaptability over a wide dynamic range makes it a desirable feature.” Tong, 273. Tong states that Tong’s experimental results show that “programs such as speech recognition and image processing use significantly less power with our reduced bitwidth FP representation than with an IEEE-standard FP representation.” Tong, 273. Tong

also notes that “[i]t has long been known that many such signal processing applications can get by with less precision/range than full FP,” *i.e.*, than with the an “IEEE-standard FP representation.” Tong, 273. Tong notes that “multiplication is one of the most frequent operations in signal processing applications,” and that “the multiplier is by far the most power hungry arithmetic unit” in a floating-point unit. Tong, 274. Of the various components in the multiplier, Tong explains that “the mantissa multiplier always consumes more power than all other blocks.” Tong, 275. Because “the mantissa multiplier itself dominates the power in an FP multiplier,” Tong teaches that “[t]o reduce power dissipation, we should focus our efforts here.” Tong, 275-276. Tong then describes experiments that show that “reduction in the mantissa bitwidth is the most effective means of reducing power dissipation.” Tong, 277.

327. As I explain further in paragraphs 328-331 below, Tong teaches how to determine, by “emulat[ing] in software different bitwidth FP units” and “plot[ting] the accuracy” of various signal-processing applications “across a range of mantissa bitwidths” (Tong, 278), “the *minimal* number of mantissa... bits” that can be used in each application “to *reduce* power consumption, yet *maintain* [the] program’s overall accuracy” (Tong, 273, emphasis original). As I also explain further in paragraphs 328-331 below, Tong demonstrates that in the “Sphinx” speech recognition application and the “ALVINN” navigation application, “the

accuracy does not change significantly with as few as 5 mantissa [fraction] bits.”

Tong, 278-279, 273, 282, Fig. 6.

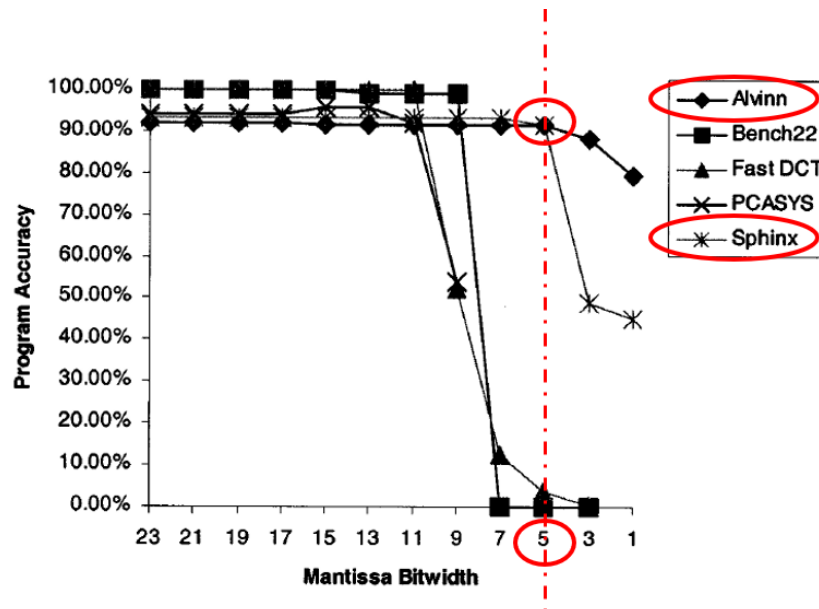


Fig. 6. Program accuracy across various mantissa bitwidths.

Tong, Fig. 6 (annotated)

328. In its Introduction, Tong states that its authors “examine how software can employ the *minimal* number of mantissa and exponent bits in FP [floating-point] hardware to *reduce* power consumption, yet *maintain* a program’s overall accuracy.” Tong, 273 (emphasis in original).

329. “To study the relationship between program accuracy and number of bits in FP [floating-point] representation,” Tong analyzes “a set of five signal processing applications.” Tong, 278. “To determine the impact of different

mantissa and exponent bitwidths,” Tong’s authors “emulated in software different bitwidth FP units by replacing each FP operation with a corresponding function call to our FP software emulation package that initially implements the IEEE-754 standard.” Tong, 278.

330. In its “Results” section, Tong explains that Figure 6 “plots the accuracy for each of the five programs across a range of mantissa bitwidths,” and explains that the results shown in Figure 6 demonstrate that for the “ALVINN and Sphinx III” programs, “the accuracy does not change significantly with as few as 5 mantissa bits;”—*i.e.*, 5 fraction bits, as I explain in paragraph 325 above. Tong, 278; *see also* Tong, 282 (“Both Sphinx and ALVINN need only 5 bits of mantissa to be 90% accurate....”). Tong also explains that Figure 7 “shows that each program’s accuracy has a similar trend when exponent bitwidth is varied.” Tong, 278. A POSA would thus have understood that Tong’s Figures 6 and 7 describe the results of two different experiments; one (Fig. 6) that “initially implements the IEEE-754 standard” but then varies the *mantissa* size (by changing the number of fraction bits), and another (Fig. 7) that “initially implements the IEEE-754 standard” and then varies the *exponent* size. Tong, 278-279.

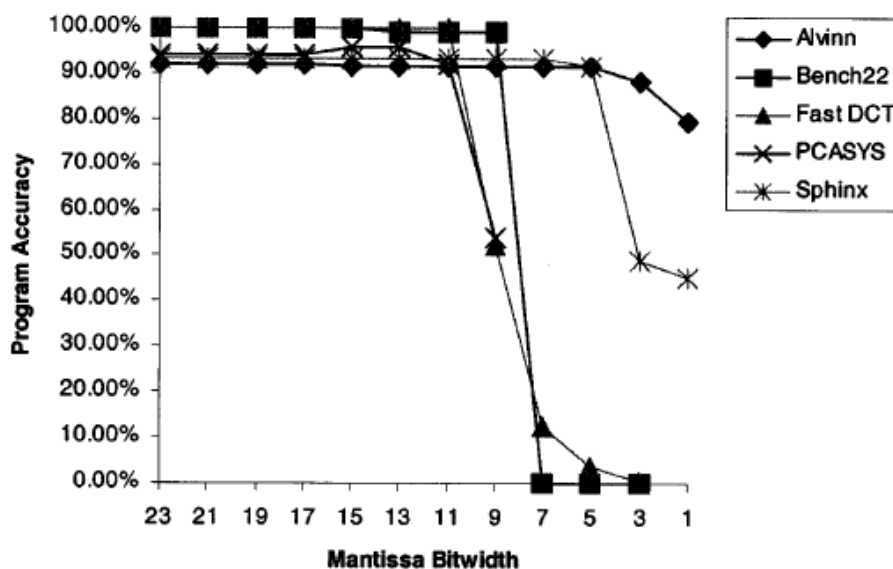


Fig. 6. Program accuracy across various mantissa bitwidths.

331. Tong explains that the reason some applications can run accurately with fewer mantissa bits is because:

many *programs dealing with human interfaces* process sensory data with intrinsically low resolutions. Raw input *data with 4–10 bits of precision is rather common* in these applications. While *intermediate results often require more dynamic range* than is available with small bitwidth fixed point computation, *the programs do not require vastly more precision*. This is different from scientific programs such as large-scale computational fluid dynamics or electrical circuit simulation, which not only require a huge amount of precision and dynamic range but also delicate rounding modes to preserve the accuracy of the results. *What is quite clear from these experiments is that the FP format provides essential dynamic range* (we can reduce, but not reduce dramatically, the number of exponent bits) *but the fine*

precision of the 23-bit mantissa is not essential (half as many bits often suffice).

Tong, 278-279.

332. Having empirically determined the minimum number of mantissa bits necessary to maintain acceptable accuracy of particular applications (*e.g.*, ALVINN and Sphinx), Tong teaches using “reduced-precision arithmetic” to “omit the unnecessary bits and computations on them” (Tong, 284), to thereby “reduce waste” (Tong, 273) and “significantly reduce... power consumption while maintaining a program’s overall accuracy” (Tong, 274). Tong, 279 (floating-point “provides essential dynamic range... but the fine precision... is not essential”). Tong’s “fundamental principle is to reduce waste—in this case, unnecessary bits in the FP representation and computation.” Tong, 273. Tong notes that “[f]or an increasing number of embedded applications such as voice recognition, vision/image processing, and other human-sensory-oriented signal-processing applications, [floating point’s] simplified programming model ... and large dynamic range makes FP hardware a desirable feature.” Tong, 274. Tong also notes that “many recognition applications achieve a high degree of accuracy starting from fairly low-resolution input sensory data.” Tong, 274. Tong explains that by “[l]everaging these characteristics by allowing software to use the minimal number of mantissa and exponent bits, standard FP hardware can be modified to

significantly reduce its power consumption while maintaining a program’s overall accuracy.” Tong, 274. Tong’s “experiments,” which Tong describes in Section V, show that “the FP format provides essential dynamic range ... but the fine precision of the 23-bit mantissa is not essential.” Tong, 279. Tong recommends reducing precision when appropriate by “omit[ting] the unnecessary bits and computations on them.” Tong, 284.

B. Claims 1-2, 21-24, 26, and 28

333. Tong’s teachings and empirical tests, which I discussed in Section VII.A above, would have motivated a POSA to configure Dockser’s FPP to operate at the precision levels Tong teaches for particular applications. Tong teaches that “[m]any mobile/portable electronics applications” perform signal processing programs “such as speech and image recognition” that are “better served with a custom, reduced[-precision] FP format.” Tong, 284 (“Many mobile/portable electronics applications will ultimately need to process human-sensory data such as speech or video imagery. ... The intuition underlying this study is that many of these applications—especially those that process low-resolution data and render as a result a decision, such as speech and image recognition—may be better served with a custom, reduced FP format.”). A POSA would have understood Dockser’s FPP with its selectable precision level for “[p]ower management” in “battery operated devices where power comes at a

premium” (Dockser, [0003]) to be well-suited for use in the “mobile/portable” applications that Tong refers to.

334. As I explained in Section VII.A above, Tong’s test results showed that for certain types of applications, such as Sphinx, ALVINN, and other “programs dealing with human interfaces,” (Tong, p. 278), the full 23-fraction-bit precision of the IEEE 754 single format is not required, and 5 bits are sufficient. In further view of Dockser’s teaching that using more precision than necessary for an application wastes power (*see* Dockser, [0003] and my discussion of Dockser in Section VI.A above), Tong’s teaching that a precision level retaining 5 mantissa fraction bits is sufficient in some applications (including ALVINN and Sphinx) would have motivated a POSA to configure Dockser’s FPP in the device I discussed in Section VI above to operate at a selected precision level retaining as few as 5 mantissa fraction bits to conserve power when running those applications, or others empirically determined (using Tong’s techniques) to not require greater precision.

335. The resulting “Dockser/Tong” device meets claims 1-2, 21-24, 26, and 28 of the ’273 patent for the same reasons I discussed in Section VI above with respect to Dockser, with the addition that Dockser/Tong’s device retaining 5 fraction bits exceeds claim 1’s X and Y values (X=5%, Y=0.05%) by greater

margins than Dockser's device retaining 9 fraction bits (*see* Section VI), as I explain in the following paragraph.

336. My software program (which I discussed in Sections VI.B.4.c(1)(i) and VI.B.4.c(2) above with respect to limitation [1B2], and which I explain in detail in Appendix I.B and I.D. below) that tests all possible valid input pairs demonstrates that when retaining 5 mantissa fraction bits in view of Tong, Dockser's register bit-dropping technique produces at least $Y=0.05\%$ relative error for 99.45% of possible valid inputs, and Dockser's logic bit-dropping technique produces at least $Y=0.05\%$ relative error for 99.47% of possible valid inputs. Additionally, the algebraic analysis I discussed in Section VI.B.4.c(1)(ii) above and in Appendix I.C. below also demonstrates that Dockser's register bit-dropping technique when retaining 5 mantissa fraction bits produces a minimum of 1.56% relative error (greater than $Y=0.05\%$) for over 12% of possible valid inputs (greater than $X=5\%$). Thus Dockser/Tong meets the same claims of the '273 patent as Dockser does.

337. Moreover, Tong and Dockser both demonstrate that POSAs knew how to optimize the number of mantissa bits as a result-effective variable for concurrently reducing precision and power consumption in a floating-point execution unit, and how to empirically determine the optimum number of bits for a particular application by testing the application's accuracy at various mantissa

bitwidths. *See, e.g.*, Tong, Section V.A, p. 278 (describing “Workloads and Methodology” to “determine the impact of different mantissa and exponent bitwidths”); Dockser, [0003] (“some graphics applications may only require a 16-bit mantissa”), [0026] (“if ... the subprecision required for the floating-point operation is 10-bits, only the 9 commonly significant bits (MSBs) of the fraction are required”). *See* my discussions of Dockser’s and Tong’s result-effective variable teachings in Section VI above and Section VIII.G below.

338. The ’273 patent lists six possible X percentages (ranging from 1% to 50%) and nine possible Y percentages (ranging from 0.05% to 20%), which together make fifty-six possible combinations of X and Y values, as “merely examples [that] do not constitute limitations of the... invention,” with no indication that any particular claimed X-Y combination is critical in any way. ’273 patent, 27:41-62 (“for at least fraction F of the possible valid inputs to that operation ... the statistical mean, over repeated execution, of the numerical values represented by that output signal ... differs by at least E from the result of an exact mathematical calculation of the operation on those same input values, where F is 1% and E is 0.05%. In several other example embodiments, F is not 1% but instead is one of 2%, or 5%, or 10%, or 20%, or 50%. For each of these example embodiments, each with some specific value for F, there are other example embodiments in which E is not 0.05% but instead is 0.1%, or 0.2%, or 0.5%, or

1%, or 2%, or 5%, or 10%, or 20%. These varied embodiments are merely examples and do not constitute limitations of the present invention.”).

339. Based on a POSA’s knowledge, which I am familiar with as I discussed in paragraph 44 above, and based on Dockser’s and Tong’s teachings, determining the optimum range of imprecision to achieve the best power reduction without sacrificing accuracy for a particular application was a matter of routine optimization of a result-effective variable, such that arriving at imprecisions resulting in the device meeting the claimed X and Y values would have been obvious.

C. Claim 32: “The device of claim 1, wherein the device is adapted to perform nearest neighbor search.”

340. As I discussed in Section VII.B above, Dockser/Tong implements Dockser’s FPP using the precision Tong teaches for neural-network image-classification applications including ALVINN. A POSA would have had reason to implement Dockser/Tong to “perform nearest neighbor search” as claim 32 of the ’273 patent recites, because a POSA would have understood that nearest-neighbor search was commonly used and well known to be advantageous for image-classification (including neural-network) applications. For evidence corroborating that this was background knowledge for a POSA, *see, e.g.*, Lang (Ex. 1056), [0001]-[0004]—*e.g.*, [0001] (“The present invention relates generally to the field of data processing, and more particularly, to methods and apparatus for

incremental approximate nearest neighbor searches in large data sets.”), [0002] (“Nearest neighbor searching is an important problem in various applications, including ... imagine [*sic*: image] retrieval ... [and] pattern recognition...”), [0003]-[0004] (explaining that “[i]n many applications, users are satisfied with finding an approximate answer that is ‘close enough’ to the exact answer” and describing algorithm for nearest-neighbor search); Tremiolles (Ex. 1057), 1:15-50 (*e.g.*, “Detecting anomalies such as abnormalities or singularities in **images**, signals, and sets of data is an every day occurrence for some professions ... Artificial **neural networks** appear to be perfectly adapted to handle this problem because they present the essential advantage of real time processing (particularly when implemented in hardware) and adaptation. Before going further, it is important to understand the principles that are at the base of artificial neural networks. To date, some artificial **neural networks** are hardware implementations of the Region Of Influence (ROI) and/or the K **Nearest Neighbor** (KNN) algorithms. ... The logical decision unit determines either the neuron which fired or the neuron which is the nearest neighbor (closest prototype) depending upon the algorithm being used. The output layer returns the appropriate **categories** (i.e. the categories associated to the neurons selected by the logical decision unit).”).

D. Claim 33

341. Tong teaches to “emulate[] in software different bitwidth FP units” “to determine application accuracy.” Tong, 278 (“To determine the impact of different mantissa and exponent bitwidths, we emulated in software different bitwidth FP units by replacing each FP operation with a corresponding function call to our FP software emulation package that initially implements the IEEE-754 standard (Fig. 5). Careful modifications to the FP emulation package allowed us to emulate different mantissa and exponent bitwidths. Then, each program was run using the modified FP package, and the results were compared to determine application accuracy.”). A POSA would have understood that the Dockser/Tong device I discussed in Section VII.B above is a kind of “FP unit” as Tong describes, since that device is the FPP of Dockser configured to use a particular number of mantissa bits. *See* Dockser, [0001] (“Floating-point processors are specialized computing **units** that perform certain mathematical operations ...”).

342. Tong’s teaching to “emulate[] in software different bitwidth FP units” “to determine application accuracy” (Tong, 278) would have motivated a POSA to emulate the Dockser/Tong device I discussed in Section VII.B above in software to assess the accuracy of the applications running on the device at selected precision levels. A POSA would have had a reasonable expectation of success in doing so, because writing software that performed reduced-precision multiplication using

Dockser’s techniques with the particular number of fraction bits taught by Tong would have been a straightforward task for a POSA. This can be seen, for example, from the source code for my own testing programs (which is attached to this Declaration as Attachments A1-B2); the code performs reduced-precision multiplication using basic instructions in the C programming language that a POSA would have known how to use. Indeed, Dockser teaches that the FPP “methods or algorithms... may be embodied... in a software module.” Dockser, [0036].

1. **[33A1] “A device comprising a computer processor and a computer-readable memory storing computer program instructions, wherein the computer program instructions are executable by the processor to emulate a second device comprising: a plurality of components comprising: at least one first low precision high-dynamic range (LPHDR) execution unit”**

343. A POSA would have understood that when the Dockser/Tong device is “emulated in software” per Tong’s teachings (Tong, 278), which I discussed in Section VII.A above, the conventional and obvious implementation of such “software” is “computer program instructions” stored in “a computer-readable memory” and “executable by [a] processor” within a “device” (e.g., a computer) as limitation [33A1] recites. For evidence corroborating that this is what a POSA would have understood based on a POSA’s background knowledge, *see, e.g.*, Dockser, [0036] (“software” is “executed by a processor” and “reside[s] in [a]

memory”); Donovan (Ex. 1029), [0008] (“From the standpoint of the computer’s hardware, most systems operate in fundamentally the same manner. Processors are capable of performing very simple operations ... Sophisticated software at multiple levels directs a computer to perform massive numbers of these simple operations, enabling the computer to perform complex tasks.”), [0009] (“computer, computer programs which instructed the computer to perform some task were written in a form directly executable by the computer’s processor ... alternative forms of creating and executing computer software were developed. In particular, a large and varied set of high-level languages was developed for supporting the creation of computer software”), [0014] (“software” is “computer program with instructions”). Furthermore, the ’273 patent states that a “software emulator” is an “embodiment[] of the [claimed] invention”. ’273 patent, 26:2-4.

344. As I discussed in Section VII.D above, a POSA would have been motivated by Tong’s teachings to use a computer (which is a claimed “device comprising a computer processor...[etc.]” as I discussed in paragraph 343 above) to emulate the Dockser/Tong device I discussed in Section VII.B above. This emulated device comprises Dockser’s LPHDR execution unit, thereby meeting the “second device” of limitation [33A1]. *See* Section VII.D.2 below (where I discuss how Dockser’s execution unit in the Dockser/Tong combination meets claim 33’s

recited limitations of the LPHDR execution unit, which are identical to those recited in claim 1).

345. Like the '273 patent's only described examples of the claimed "emulat[ing] a... device" comprising LPHDR execution unit(s), Tong emulates the device by executing a software application program that would run on the device being emulated, with arithmetic operations in the application program replaced by software simulations of the reduced-precision arithmetic that the device would perform while running that application program, as I explain below and in the following paragraph. *See* '273 patent, 17:3-17, 18:52-19:12; Tong, 278. The '273 patent describes an experiment in which "[a]pplications are tested using two embodiments for the machine's arithmetic," one that "represent[s] the results produced by an analog embodiment of the machine," and another that "uses logarithmic arithmetic," which "represent[s] the results produced by an analog embodiment of the machine." '273 patent, 17:3-17. These tests "express[] the arithmetic performed ... as code in the C programming language." '273 patent, 18:52-55. This code "contains several implementations for arithmetic." '273 patent, 19:4-5. As the patent explains:

When compiled with "#define fp" the arithmetic is done using IEEE standard floating point. If a command line argument is passed in to enable noisy arithmetic, then random noise is added to the result of every calculation. This is the "fp+noise" form of arithmetic. When

compiled without “#define fp” the arithmetic is done using low precision logarithmic arithmetic with a 6 bit base-2 fraction. This is the “l ns” form of arithmetic.

'273 patent, 19:5-12. Based on this description, a POSA would have understood that the tests described in the '273 patent involve compiling the software that performs the tests in such a way that when the software performs certain floating-point arithmetic operations, the software uses either the “fp+noise” or “l ns” forms of floating-point arithmetic.

346. In the same fashion as the '273 patent's emulations, Tong “emulate[s] in software different bitwidth FP units by replacing each FP operation with a corresponding function call to our FP software emulation package that initially implements the IEEE-754 standard (Fig. 5).” Tong, 278. This is illustrated in Tong's Figure 5, which shows that when the “Original Code” is “Annotated with calls to emulator,” the traditional addition, subtraction, and multiplication operations are replaced with “fadd,” “fsub,” and “fmult” operations, respectively.

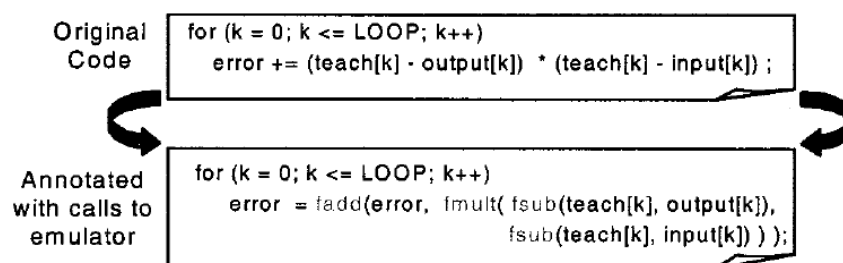


Fig. 5. FP emulation by annotating the source code.

Tong, Fig. 5

2. Limitations [33A2]-[33B2]

347. As shown below, limitations [33A2]-[33B2] are identical to limitations [1A2]-[1B2], respectively.

Claim 33 limitations	Claim 1 limitations
[33A2] adapted to execute a first operation on a first input signal representing a first numerical value to produce a first output signal representing a second numerical value;	[1A2] adapted to execute a first operation on a first input signal representing a first numerical value to produce a first output signal representing a second numerical value,
[33B1] wherein the dynamic range of the possible valid inputs to the first operation is at least as wide as from 1/65,000 through 65,000 and	[1B1] wherein the dynamic range of the possible valid inputs to the first operation is at least as wide as from 1/65,000 through 65,000 and
[33B2] for at least X=5% of the possible valid inputs to the first operation, the statistical mean, over repeated execution of the first operation on each specific input from the at least X % of the possible valid inputs to the first operation, of the numerical values represented by the first output signal of the LPHDR unit executing the first operation on that input differs by at least Y=0.05% from the result of an exact mathematical calculation of the first operation on the numerical values of that same input.	[1B2] for at least X=5% of the possible valid inputs to the first operation, the statistical mean, over repeated execution of the first operation on each specific input from the at least X % of the possible valid inputs to the first operation, of the numerical values represented by the first output signal of the LPHDR unit executing the first operation on that input differs by at least Y=0.05% from the result of an exact mathematical calculation of the first operation on the numerical values of that same input.

348. Therefore, because the emulated “second device” in Dockser/Tong is the device discussed in Sections VI.B and VII.B above, Dockser/Tong meets limitations [33A2]-[33B2] for the same reasons Dockser’s device (operating at

Dockser’s or Tong’s disclosed precision levels) meets the identical limitations in claim 1.

VIII. CLAIMS 1-26, 28, 36-61, AND 63 WOULD HAVE BEEN OBVIOUS OVER DOCKSER AND MACMILLAN

A. MacMillan (Ex. 1009)

349. MacMillan “improv[es]... speed of a personal computer architecture through... parallel processing capability” “[t]o provide supercomputer performance in a computer for personal use.” MacMillan, 8:38-40, 5:22-45, 1:10-47. MacMillan states in its “Background of the Invention” section that “[i]f supercomputing performance could be achieved in low cost computers, such as personal computers, this could dramatically expand the market for personal computers.” MacMillan, 1:20-23. MacMillan teaches that because of performance limitations of “uniprocessor-based” systems, “a personal computer providing supercomputer performance should use a parallel computing architecture.” MacMillan, 1:38-47. MacMillan notes that “SIMD architectures have some qualities that make them attractive candidates for adding supercomputer performance to a computer for personal use,” but that “present SIMD architectures are much higher cost than computer systems for personal use.” MacMillan, 5:22-27. MacMillan identifies certain things that must be achieved in a SIMD system “[t]o provide supercomputer performance in a computer for personal use,” MacMillan, 5:27-28, and teaches an invention that “addresses [those] needs.”

MacMillan, 5:44-45. Specifically, MacMillan’s invention “relates to an improvement in the overall speed of a personal computer architecture through the addition of a parallel processing capability.” MacMillan, 8:38-40.

350. Consistent with how the term “supercomputer” was used in the art, MacMillan describes “supercomputer” performance in terms of speed (e.g., number of operations performed per second), not precision. *See* MacMillan, 1:15-54 (“Personal computers (PCs) today typically have performance up to about 300 million instructions per second (MIPs). This is much less than supercomputers, which may provide performance of 5,000 MIPs or more. ... Parallel computers with over 5,000 MIPs performance are readily achievable, as demonstrated by their availability from a number of vendors, although at high prices.”). For corroborating evidence of how the term “supercomputer” was used in the art, *see, e.g., Penguin* (Ex. 1052), 1411 (“any of a class of very powerful computers that are capable of performing at great speed and are used for repeated calculation cycles on vast amounts of data”); *Oxford* (Ex. 1053), 500 (“supercomputer” is a “class of very powerful computers” capable of “performing several hundred [trillion] floating-point operations per second”).

351. MacMillan achieves “substantial performance improvements” by adding a “Single Instruction Multiple Data (SIMD)” subsystem to “existing system architectures.” MacMillan, 9:17-19, 5:48-6:10. MacMillan’s invention

“comprises a computer system to which has been added a Single Instruction Multiple Data (SIMD) parallel processing capability.” MacMillan, 5:48-50. “The computer system, to which the SIMD capability has been added, could be a computer system for personal use or other computer system.” MacMillan, 5:50-53. MacMillan’s “cost-effective addition of parallel processing can be made to popular, low cost computer systems running popular operating systems,” including “Microsoft Windows based personal computers” and “UNIX-based systems.” MacMillan, 6:3-10. MacMillan provides this “SIMD capability” with a “SIMD subsystem” that “adds parallel processing capabilities to the personal computer, resulting in substantial performance improvements.” MacMillan, 9:17-19.

352. MacMillan’s SIMD subsystem includes SIMD-RAM devices having multiple parallel “processing elements (PE’s),” each PE including “floating point accelerators” that “perform... operations on... 32-bit words.” MacMillan, 9:11-19, 12:35-59, FIG. 2, 5. Figure 2 of MacMillan shows a “SIMD subsystem 100” that “includes a SIMD Controller” and “one or more SIMD-RAM devices 104.” MacMillan, 9:11-12. “Fig[ure] 5 shows a block diagram of a SIMD-RAM device.” MacMillan, 12:35-36. The SIMD-RAM contains “a plurality of memory devices, in this case, dynamic random access memory (DRAM) 304, which are coupled to a plurality of processing elements (PE's) 302 via random access logic 306.” MacMillan, 12:36-40. “Each PE contains a 32-bit wide data path and can

perform atomic operations on bits, bytes, 16-bit words, and 32-bit words.”

MacMillan, 12:47-49. In addition, “[i]nteger and floating point accelerators could be included in each PE.” MacMillan, 12:55-56.

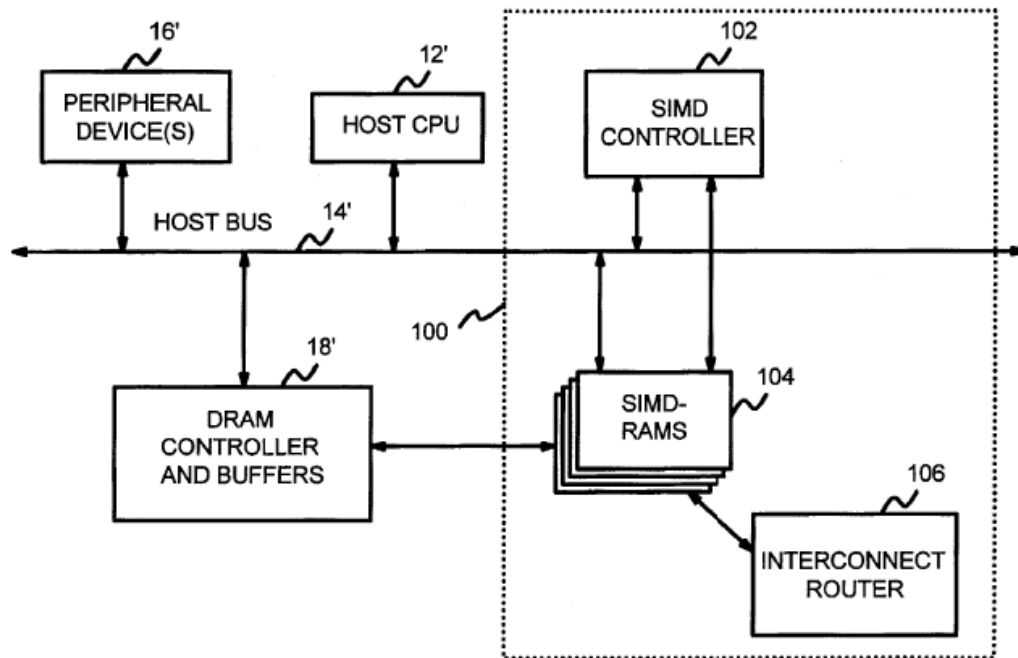
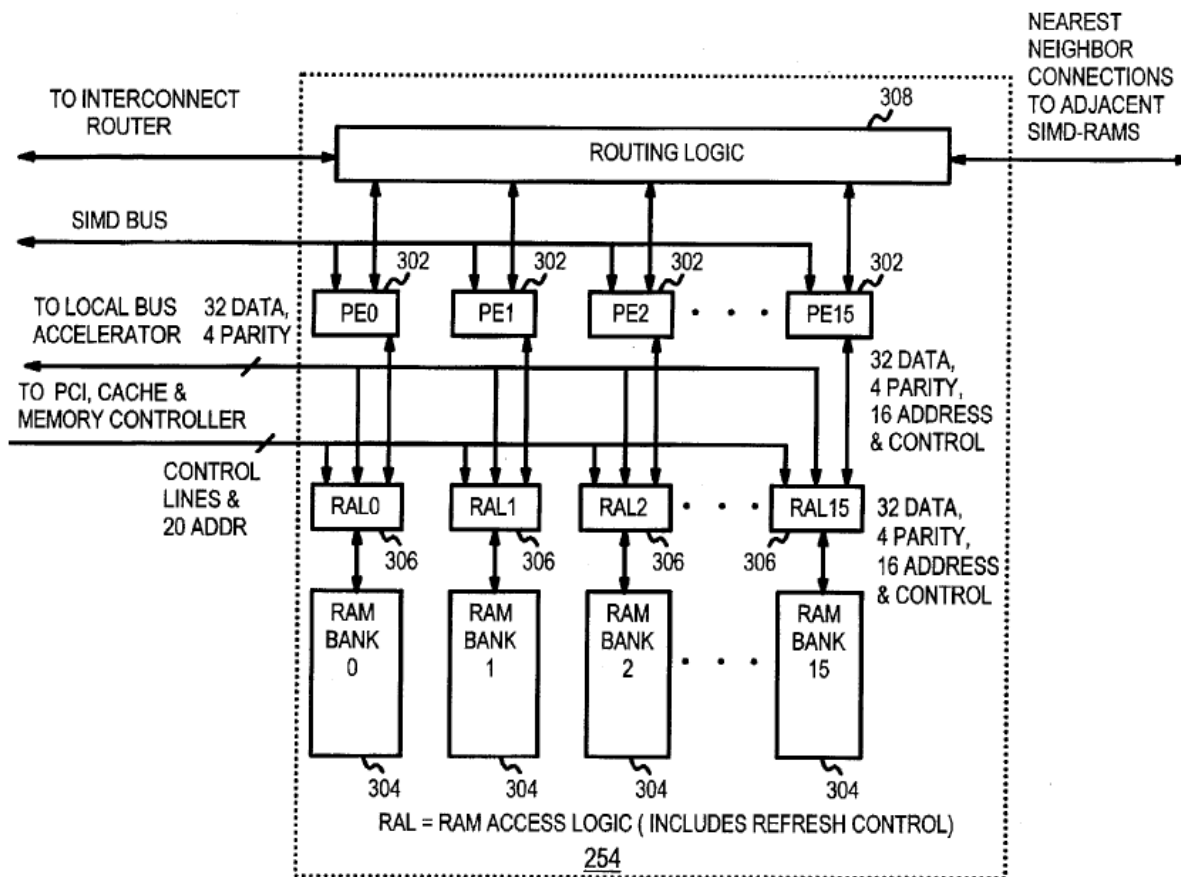


FIG. 2

MacMillan, Fig. 2 (SIMD system)



MacMillan, Fig. 5 (SIMD-RAM device)

B. Dockser/MacMillan Combination

353. MacMillan teaches incorporating SIMD parallel processing in personal-use computers, including battery-operated devices like “laptops, palmtops, [and] personal digital assistants”—the same types of computers for which Dockser’s FPP provides beneficial “[p]ower management” through reduced-precision floating-point operations. MacMillan, 1:6-9, 5:22-45, 7:14-34; Dockser, [0003]. MacMillan’s invention relates to “computer systems for personal use.”

MacMillan, 1:6-9. As MacMillan explains, “SIMD architectures have some qualities that make them attractive candidates for adding supercomputer performance to a computer for personal use,” but existing “SIMD architectures” have “limitations.” 5:22-26. “Overcoming” those limitations, MacMillan explains, “would allow supercomputing performance to be provided in a low cost computer system for personal use,” and MacMillan’s invention “addresses” those “needs”—*i.e.*, the needs to overcome those limitations. MacMillan, 5:42-45.

MacMillan defines “[a] computer system for personal use” as

a system that is designed primarily for use by an individual or for a specific application, including embedded applications. ***A computer system for personal use includes*** personal computers (including, but not limited to, those that are IBM-compatible and Apple-compatible), workstations (including, but not limited to, those that use RISC technology or that use the UNIX operating system), embedded computers (including, but not limited to, computer-based equipment used for telecommunications, data communications, industrial control, process control, printing, settop boxes, signal processing, data compression or decompression, data transformation, data aggregation, data extrapolation, data deduction, data induction, video or audio editing or special effects, instrumentation, data collection or analysis or display, display terminals or screens, voice recognition, voice processing, voice synthesis, voice reproduction, data recording and playback, music synthesis, animation, or rendering), ***laptops***,

palmtops, personal digital assistants, notebooks, subnotebooks, and video games.

MacMillan, 7:14-34. These are the same types of systems in which, according to Dockser, the problem of “unnecessary power consumption” is “of particular concern.” Dockser, [0003] (“some graphics applications may only require a 16-bit mantissa. For these graphics applications, any accuracy beyond 16 bits of precision tends to result in unnecessary power consumption. This is of particular concern in battery operated devices where power comes at a premium, such as wireless telephones, personal digital assistants (PDA), laptops, game consoles, pagers, and cameras, just to name a few.”). Dockser teaches that “there is a need in the art for a floating-point processor in which the reduced precision, or subprecision, of the floating-point format is selectable,” which Dockser addresses with this selectable-precision FPP. Dockser, [0003]. Thus, both MacMillan and Dockser are directed to improvements in the same types of devices.

354. In addition, MacMillan teaches that “power dissipation and hence power supply capacity and cost” should be considered in pursuing the performance improvements of MacMillan’s SIMD architecture. MacMillan, 3:2-6. MacMillan explains that existing “SIMD architectures...suffer from a number of shortcomings.” MacMillan, 2:21-26. One of these limitations is a “bottleneck between the chips used to hold PEs [processing elements] and the chips used to

provide memory storage.” MacMillan, 2:26-48. Although MacMillan notes that “[a]dding pins can reduce the PE-to-memory bottleneck, but leads to increased packaging costs,” MacMillan also notes that “[a]dding output buffers to drive increased pin counts also increases power dissipation and hence *power supply capacity and cost*,” which in turn “may also require increased space between chips or circuit boards for better cooling, leading to larger, more costly cabinets.” MacMillan, 2:62-3:6. One of MacMillan’s goals is to “overcom[e]” the “limitations” it identifies in prior art SIMD systems.” MacMillan, 5:38-42. Thus, a POSA would have understood that one of MacMillan’s goals in the design of its SIMD system was addressing “power dissipation and hence power supply capacity and cost.” MacMillan, 3:3-4.

355. Based on MacMillan’s and Dockser’s teachings that I discuss above, a POSA would have been motivated to use Dockser’s FPP to implement each “floating-point accelerator[]” in the parallel PEs of MacMillan’s SIMD architecture (MacMillan, 12:55-56), to increase performance speed as MacMillan teaches while lowering power consumption as Dockser teaches.

356. A POSA would have understood that Dockser’s FPP is a “floating-point accelerator” for “perform[ing] atomic operations on... 32-bit words” as in MacMillan’s PE, and using Dockser’s FPP as the floating-point accelerator in MacMillan’s PE would have achieved the predictable result of enabling the PEs to

perform reduced-precision floating-point arithmetic as taught by Dockser at reduced power. MacMillan, 12:47-59. As I discussed in paragraph 352 above, MacMillan discloses that its PEs “can perform atomic operations on bits, bytes, 16-bit words, and 32-bit words.” MacMillan, 12:47-49. MacMillan also teaches including “floating-point accelerators” in each PE. MacMillan, 12:55-56. As a POSA would have understood, a floating point accelerator is a processor element for executing floating point operations, as is Dockser’s FPP. For evidence corroborating that this was background knowledge for a POSA, *see, e.g.*, Fossum (Ex. 1024), 1:43-58 (explaining in the “Description of the Prior Art” that in “Data processing (i.e. computer) systems,” “[s]ometimes a special type of *processor element*, termed a *floating point accelerator*, is provided for performing floating point arithmetic calculations. Floating point accelerators are specifically designed to increase the speed with which floating point calculations may be performed; when a *floating point operation* is to be performed, it is *executed* in or by the floating point accelerator rather than in another processor.”). Thus, a POSA would have understood that the “floating-point accelerators” in the PEs in MacMillan are processor elements that perform floating-point operations, just like Dockser’s FPP, and therefore would have understood that using Dockser’s FPP as the floating-point accelerator would enable MacMillan’s PEs to perform reduced-precision floating-point arithmetic.

357. Moreover, MacMillan teaches that its SIMD architecture is beneficial for applications including “animation,... rendering..., and video games,” which a POSA would have understood are types of the “graphics applications” for which Dockser teaches that its FPP can beneficially save power by reducing unnecessary precision. MacMillan, 7:14-33; Dockser, [0003]. As I discussed in paragraph 353 above, MacMillan’s primary focus is on “computer system[s] for personal use,” which includes “laptops, palmtops, personal digital assistants, notebooks, subnotebooks, and video games,” MacMillan, 7:14-34, which are the same types of “battery operated devices where power comes at a premium” that Dockser is primarily focused on. *See* Dockser, [0003] (“wireless telephones, personal digital assistants (PDA), laptops, game consoles, pagers, and cameras, just to name a few”).

C. Claims 1 and 28

358. In the combination of Dockser and MacMillan (“Dockser/MacMillan”) discussed in Section VIII.B above, MacMillan’s “computer system” (*e.g.*, computer system 200) (MacMillan, 7:14-34, 9:20-29), or alternatively a “SIMD-RAM *device*” (element 104/254) within MacMillan’s computer system (MacMillan, 9:11-29, 12:35-59) meets the “device” of limitation [1A1], and comprises Dockser’s execution unit, as I explain in the following paragraph.

359. As I explained in paragraph 353 above, MacMillan teaches improvements to “computer *system[s]* for personal use.” MacMillan, 7:14. *See also* MacMillan, 9:20-31 (“An additional embodiment of the present invention is shown in FIG. 3. This includes a personal computer system 200, which includes a SIMD subsystem 250, in accordance with the present invention.”). A POSA would have understood that MacMillan’s “system” (MacMillan, 7:14-34, 9:20-31) is a claimed *device*. *See, e.g.*, ’273 patent, 1:61 (“Real computers are built as physical devices...”), 24:10-11 (referring to “conventional computing devices”), 27:63-66 (“For certain *devices (such as computers or processors or other devices)* *embodied according the present invention*, the number of LPHDR arithmetic elements in the device (e.g., *computer or processor or other device*) exceeds...”), 7:40-43 (“Various computing *devices* implemented according to embodiments of the present invention will now be described. Some of these embodiments may be an instance of a *SIMD computer* architecture. “), 29:5-16 (“Embodiments of the present invention may, however, be implemented in devices in addition to or other than processors ...More generally, any device or combination of devices, whether or not falling within the meaning of a ‘processor,’ which performs the functions disclosed herein may constitute an example of an embodiment of the present invention.”). Dockser/MacMillan’s computer system comprises at least one SIMD-RAM (which as MacMillan describes is a “device”) containing PEs

(processing elements), each of which contains one or more Dockser FPPs as “floating-point accelerators,” as I explained in Sections VIII.A-VIII.B above. MacMillan, 8:56-9:31, 12:55-56.

360. All other limitations of claims 1 and 28 of the ’273 patent relate to characteristics of the LPHDR execution unit, which in Dockser/MacMillan is the same Dockser execution unit I discussed in Section VI above. Thus, Dockser/MacMillan meets the remaining limitations in claims 1 and 28 for the reasons discussed in Sections VI.B and VI.I. above.

D. Claims 3, 7, and 9

361. MacMillan discloses an example architecture including 256 PEs, and discloses scaling that architecture to reach “tens of thousands of processors [PEs] or more.” MacMillan, 12:60-13:4, 13:39-41, 16:20-22, 2:13-16. MacMillan discloses “example[s]” of systems with 256 PEs, divided among different numbers of SIMD-RAMs. MacMillan, 12:66-13:4 (“an application program would see no difference, *for example*, between ... sixteen SIMD-RAM 254 chips, each having 1 Mbyte DRAM and 16 SIMD PEs, and ... a single SIMD-RAM 254 chip containing 16 Mbytes of DRAM and 256 PEs”); *see also* 13:39-41 (“FIG. 6 compares the address map of the Host CPU 208 to that of the PEs 302 for a system with 256 PE's (as might be provided by sixteen SIMD-RAM chips, each with sixteen PEs).”). However, MacMillan explains that “[t]he architecture of the

SIMD-RAM” that MacMillan teaches “allows scaling to higher or lower density chips with more or fewer PEs [], more or less memory [], and different amounts of memory per PE.” MacMillan, 12:60-63. Additionally, MacMillan’s “architecture is easily scaled across a range of performance simply by adding SIMD-RAM chips.” MacMillan, 16:20-22. Furthermore, MacMillan’s system is a SIMD system, and MacMillan notes that “[t]he SIMD architecture can be scaled to tens of thousands of processors or more, with excellent processor utilization for many programs.” MacMillan, 2:13-16.

362. Thus, the Dockser/MacMillan system having at least one Dockser LPHDR execution unit in each of 256 or up to tens of thousands of PEs meets claims 3, 7, and 9 of the ’273 patent, reciting that “the at least one first LPHDR execution unit comprises at least ten LPHDR execution units” (claim 3), that “the at least one first LPHDR execution unit comprises at least one hundred LPHDR execution units” (claim 7), and that “the at least one first LPHDR execution unit comprises at least five hundred LPHDR execution units” (claim 9).

E. Claims 5, 8, and 10

363. As I discussed in Section VIII.C above (*e.g.*, paragraphs 358-360), in Dockser/MacMillan, MacMillan’s SIMD computer system meets the “device of claim 1” of the ’273 patent. MacMillan’s computer system includes a Host CPU implementable as “a 386, 486 or Pentium™ processor.” MacMillan, 9:30-31.

POSAs understood these processors included a floating-point execution unit that performed multiplication on IEEE-754 single-format 32-bit floating-point numbers, as I explain in paragraphs 364-368 below.

364. A POSA would have understood that “[m]any computer processors have adopted the IEEE Standard for Binary Floating-Point [Arithmetic] ... referred herein as ‘IEEE 754.’” Prabhu (Ex. 1017), [0005]; *see also* Dockser, [0002] (referring to “the ANSI/IEEE-754 standard (commonly followed by modern computers)”). “Examples of such processors” that adopted the IEEE 754 standard “include ... any of the Pentium or x86 compatible processors.” Prabhu, [0005]. This is consistent with my knowledge of how POSAs would have understood the state of the art in computer processors. A POSA would also have understood that processors in the x86 family typically included a single “floating-point unit” that executed floating point operations. For evidence corroborating that this was background knowledge for a POSA, *see, e.g.*, Donovan (Ex. 1029, published in 2007), [0013] (noting in its “Background Art” section that one event in the evolution of the x86 processor was “the addition of a floating-point unit to the x86 processor architecture, first as an optional co-processor, and eventually as an integrated part of every... chip”), Vitale (Ex. 1030), 1:13-19 (explaining in “Background” section that “many computer systems have special processors devoted to making floating point calculations. Typically in such a computer

system, when an instruction calls for a floating point operation, a floating point unit will perform the operation and present the result to the system (main processor.”). A POSA would have understood that this floating-point unit in the x86 processors performed floating-point arithmetic according to the IEEE 754 standard because, as discussed previously, a POSA would have understood that the x86 family had adopted the IEEE 754 standard for floating-point arithmetic. For evidence corroborating that this was background knowledge for a POSA, *see, e.g.*, Prabhu, [0005].

365. A POSA would further have understood that one of the operations that the floating-point unit in x86 processors performed in accordance with the IEEE 754 standard was multiplication. For evidence corroborating that this was background knowledge for a POSA, *see, e.g.*, Prabhu, [0005] (“Typically, graphics calculations involve multiple floating-point arithmetic operations Such as addition and multiplication. Many computer processors have adopted the IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754-1985, referred herein as ‘IEEE 754.’”). Additionally, a POSA would have understood that, under the IEEE 754 standard, the floating-point unit performed multiplication on floating-point numbers that are at least 32 bits wide, namely, single format numbers. For evidence corroborating that this was background knowledge for a POSA, *see, e.g.*,

Dockser, [0002] (“the ANSI/IEEE-754 standard (commonly followed by modern computers) specifies a 32-bit single format”).

366. A POSA would have understood that the “386” and “486” processors that MacMillan listed as options for the host CPU (MacMillan, 9:30-31) were members of the so-called “x86” family of processors from Intel Corporation. For evidence corroborating that this was background knowledge for a POSA, *see, e.g.*, Dreyer (Ex. 1018), 2:7-10 (“For example, the INTEL x86 family of microprocessors includes the 8086, the 80286, the Intel386™ (‘i386™’), and the Intel486™ (i486™) microprocessor families.”). Thus, for the reasons I discussed in paragraphs 364-365 above, a POSA would have understood that MacMillan teaches a “host CPU” with a single floating-point unit that performs multiplication on 32-bit floating point numbers.

367. A POSA would have understood that the floating point unit in MacMillan’s x86 “Host CPU” is an “execution unit.” The floating point *unit* performs, *i.e.* executes, floating-point operations. For evidence corroborating that this was background knowledge for a POSA, *see, e.g.*, Vitale (ex. 1030), 1:13-19 (explaining in “Background” section that “many computer systems have special processors devoted to making floating point calculations. Typically in such a computer system, *when an instruction calls for a floating point operation, a*

floating point unit will perform the operation and present the result to the system (main) processor.”).

368. Thus, for the reasons I discuss in paragraphs 364-367 above, a POSA would have understood that an obvious implementation of MacMillan’s CPU includes one claimed “execution unit[]...adapted to execute at least the operation of multiplication on floating point numbers that are at least 32 bits wide” (*e.g.*, a floating-point execution unit).

369. MacMillan discloses an example architecture including 256 PEs, and discloses scaling that architecture to reach “tens of thousands of processors [PEs] or more.” MacMillan, 12:60-13:4, 13:39-41, 16:20-22, 2:13-16. MacMillan discloses “example[s]” of systems with 256 PEs, divided among different numbers of SIMD-RAMs. MacMillan, 12:66-13:4 (“an application program would see no difference, *for example*, between ... sixteen SIMD-RAM 254 chips, each having 1 Mbyte DRAM and 16 SIMD PEs, and ... a single SIMD-RAM 254 chip containing 16 Mbytes of DRAM and 256 PEs”); *see also* 13:39-41 (“FIG. 6 compares the address map of the Host CPU 208 to that of the PEs 302 for a system with 256 PE's (as might be provided by sixteen SIMD-RAM chips, each with sixteen PEs).”). However, MacMillan explains that “[t]he architecture of the SIMD-RAM” that MacMillan teaches “allows scaling to higher or lower density chips with more or fewer PEs [], more or less memory [], and different amounts of

memory per PE.” MacMillan, 12:60-63. Additionally, MacMillan’s “architecture is easily scaled across a range of performance simply by adding SIMD-RAM chips.” MacMillan, 16:20-22. Furthermore, MacMillan’s system is a SIMD system, and MacMillan notes that “[t]he SIMD architecture can be scaled to tens of thousands of processors or more, with excellent processor utilization for many programs.” MacMillan, 2:13-16.

370. Thus, in the Dockser/MacMillan combination having a single Host CPU and at least one FPP in each PE (of which there are at least 256 and optionally more than “tens of thousands”), the number of FPPs (LPHDR execution units), *i.e.*, at least 256, exceeds by over 10, by over 100, and by over 500 the non-negative integer number of Host CPU floating-point execution units, *i.e.*, 1, meeting claims 5, 8, and 10 of the ’273 patent.

371. The ’273 patent states:

The degree of precision of a “***low precision, high dynamic range***” arithmetic element may vary from implementation to implementation. For example, in certain embodiments, a LPHDR arithmetic element produces results which include fractions, that is, values greater than zero and less than one. For example, ***in certain embodiments, a LPHDR arithmetic element produces results which are sometimes (or all of the time) no closer than 0.05% to the correct result*** (that is, the absolute value of the difference between the produced result and

the correct result is *no more than one-twentieth of one percent of the absolute value of the correct result*).

'273 patent, 26:50-60. The patent also states:

For certain devices (such as computers or processors or other devices) embodied according the present invention, the *number of LPHDR arithmetic elements* in the device (e.g., computer or processor or other device) *exceeds the number*, possibly zero, of *arithmetic elements in the device which are designed to perform high dynamic range arithmetic of traditional precision (that is, floating point arithmetic with a word length of 32 or more bits)*.

'273 patent, 27:63-28:3. The '273 patent thus says that “arithmetic elements... designed to perform... floating point arithmetic with a word length of 32 or more bits” are “designed to perform... arithmetic of traditional precision,” and it contrasts those traditional-precision elements with “LPHDR arithmetic elements” ('273 patent, 27:63-28:3), which may “sometimes” produce results that differ from the correct result (*id.*, 26:50-60).

372. A POSA would therefore have understood the '273 patent to describe that the claimed “execution units... adapted to execute... multiplication on floating point numbers that are at least 32 bits wide” in claims 5, 8, and 10 are “traditional precision” execution units that do not “sometimes” produce results different from the correct traditional-precision result.

373. A POSA would have understood that the floating-point execution unit of MacMillan's Host CPU is such a "traditional precision" execution unit, and would have understood that Dockser's execution unit that sometimes produces differing results from those produced by a "traditional precision" floating point unit (as I explained in Section VI.B.4 above when discussing limitation [1B2]) is a claimed "LPHDR execution unit."

374. Thus a POSA would have understood that Dockser/MacMillan meets claims 5, 8, and 10 of the '273 patent because Dockser/MacMillan's number (256 or more, up to tens of thousands) of Dockser's LPHDR execution units exceeds by over 10, over 100, and over 500 its number (one) of traditional-precision execution units (the single Host CPU floating-point unit).

F. Claims 2, 4, and 6

375. As I explained in Section VIII.B above, the Dockser/MacMillan combination includes multiple of Dockser's LPHDR execution units. One obvious implementation of Dockser/MacMillan includes an FPGA implementing each LPHDR execution unit (as Dockser [0035] teaches), so each execution unit comprises "at least part of an FPGA" as I discussed in Section VI.C above. *See* Dockser, [0035] ("The various illustrative logical units, blocks, modules, circuits, elements, and/or components described in connection with the embodiments disclosed herein may be implemented or performed in a floating-point processor

that is part of ... a *field programmable gate array (FPGA)* ...”). Thus, Dockser/MacMillan meets each of claims 2, 4, and 6 of the ’273 patent, which recite that “the at least one first LPHDR execution unit” (of claims 1, 3, and 5, respectively) “comprises at least part of an FPGA.”

376. If claims 2, 4, and 6 of the ’273 patent were alternatively interpreted to require multiple LPHDR execution units implemented in a single FPGA, that also would have been an obvious implementation of Dockser/MacMillan because it was well-known for an “FPGA [to] include multiple processing elements” like Dockser/MacMillan’s PEs, to decrease “space required for the circuit, and consequently, the circuit’s cost and... energy losses,” as I explain in the following three paragraphs. Young (Ex. 1058), [0023]-[0024].

377. Young (published in 2005) explains that “[i]n *existing* applications, single-chip FPGAs are typically reconfigured on a chip-wide basis,” and further explains that “[e]ach FPGA may include multiple processing elements, *i.e.*, a set of logic gates that are logically separated from the remaining logic gates on the FPGA.” Young, [0023]; *see also* Young, [0056] (“in one embodiment, a processing element is a portion of a chip that is logically separated from other portions of the chip, *i.e.*, the logic in a processing element doesn't directly interact with the logic in other processing elements. A reconfigurable unit may include one or more processing elements... In [an] embodiment [of Figure 9], a one to one

relationship between reconfigurable units and processing elements does not exist, *i.e.*, each reconfigurable unit includes more than one processing element.”), [0057] (describing multiple “processing element[s]” or “PE[s]” in Figure 9, each of which “process...data”), [0061]-[0064] (describing similar system with “reconfigurable units” partitioned into multiple processing elements), [0069]-[0070]). Based on Young’s text, a POSA would have understood that an FPP (containing an FPO) as Dockser teaches is a “processing element” as Young uses that term.

378. Young contrasts the use of FPGAs with the approach of using “multiple dedicated chips for each individual circuit that is desired,” which is “disadvantageous because it increases the space required for the circuit, and consequently, the circuit’s cost.” Yong, [0024]. Young also explains that systems in which an FPGA implements multiple processing elements are “particularly advantageous for applications that have significant digital signal processing requirements” (Young, [0069]) and are particularly useful in devices such as “a cell phone, a GPS receiver, a video recorder and image compressor, [or] a gaming terminal” (Young, [0070]).

379. Young’s explanation that it was known in the art to implement multiple processing elements on a single FPGA, and its explanation of the benefits of doing so, are consistent with a POSA’s background knowledge, which I am familiar with as discussed in paragraph 44 above. Based on a POSA’s background

knowledge, (of which Young provides corroborating evidence), a POSA would have been motivated to implement the multiple LPHDR execution units of the Dockser/MacMillan system on a single FPGA in order to, *e.g.*, decrease the circuit space, cost, and energy losses in the Dockser/MacMillan system, and would have had a reasonable expectation of success in doing so given that this was a known and conventional technique in the art.

G. Claims 11-17

380. Claims 11-17 of the '273 patent depend from claim 8 and each recite a minimum X and/or Y percentage that is higher than claim 1's, with the highest combination being "X=10% and... Y=0.2%" in claim 17.

Claim 11
The device of claim 8, wherein X=10%
Claim 12
The device of claim 8, wherein Y=0.1%.
Claim 13
The device of claim 8, wherein Y=0.15%
Claim 14
The device of claim 8, wherein Y=0.2%.
Claim 15
The device of claim 8, wherein X=10% and wherein Y=0.1%.
Claim 16
The device of claim 8, wherein X=10% and wherein Y=0.15%
Claim 17
The device of claim 8, wherein X=10% and wherein Y=0.2%.

381. My software program—which I discussed in Sections VI.B.4.c(1)(i) and VI.B.4.c(2) above and in detail in Appendix I.B and I.D below—demonstrates

that Dockser/MacMillan meets claim 17 of the '273 patent because when retaining 9 mantissa fraction bits as Dockser discloses at [0026], Dockser's register bit-dropping technique produces $Y \geq 0.2\%$ relative error for 14.6% of possible valid inputs, and Dockser's logic bit-dropping technique produces $Y \geq 0.2\%$ relative error for 85% of possible valid inputs. Dockser/MacMillan thus also meets claims 11-16, because the recited X and Y minimum values in claims 11-16 are no higher than claim 17's.

382. The algebraic analysis that I discussed in Section VI.B.4.c(1)(ii) above and in detail in Appendix I.C below also demonstrates that Dockser's register bit-dropping meets claims 11-12 of the '273 patent when retaining 9 fraction bits; that it meets claims 13-16 when retaining 8 fraction bits; and that it meets claim 17 when retaining 7 fraction bits. A POSA would have found it obvious to implement Dockser's FPP in the Dockser/MacMillan combination while retaining only 8 or 7 mantissa fraction bits for multiple reasons.

383. **First**, Dockser discloses that the FPP's precision level is "selectable" and discloses that a 9-fraction-bit selection is only a non-limiting "example." Dockser, [0003] ("there is a need in the art for a floating-point processor in which the reduced precision, or subprecision, of the floating-point format is selectable"), [0004] ("The method includes selecting a subprecision for the floating-point operation on one or more floating-point numbers, the selection of the subprecision

resulting in one or more excess bits for each of the one or more floating-point numbers.”), [0026] (“The subprecision select bits may be used to reduce the precision of the floating-point operation.... By way of *example*, if each location in the floating-point register file contains a 23-bit fraction, and the subprecision required for the floating-point operation is 10-bits, only the 9 commonly significant bits (MSBs) of the fraction are required; the hidden or integer bit makes the tenth.”), [0013] (“The detailed description set forth below ... is not intended to represent the only embodiments in which the present disclosure may be practiced.”).

384. A POSA would thus have understood Dockser to suggest selecting other precision levels “less than the maximum precision” (Dockser, claim 1), and would have understood selecting a precision level that retains 8 or 7 fraction bits (which is only one or two fewer than Dockser’s 9-bit explicit example) to be an obvious implementation of Dockser’s teachings.

385. ***Second***, other prior art taught a low-precision floating-point multiplier that retained even fewer than 7 fraction bits. For example, as I discussed in Section VII.A above, Tong (Ex. 1008) teaches retaining only 5 fraction bits.

386. ***Third***, a POSA would have understood that the number of mantissa fraction bits retained in Dockser is a result-effective variable, so that choosing to retain any number of fraction bits (including 8 bits or 7 bits) would have been

obvious in view of Dockser's teachings that the number of mantissa bits dropped/maintained was a "selectable" variable to be optimized for different applications, and that the selection impacts the FPP's results through the precision of the outputs that the FPP produces, as I explain in paragraphs 387-389 below. *See* Dockser, [0002]-[0003], [0014], [0024]-[0027].

387. Dockser explains in its Background section that the precision of operations can be adjusted by adjusting the mantissa size, and teaches that using only the level of precision necessary for a given application is beneficial because it saves power. *See* Dockser, [0002] ("The precision of the floating-point processor is defined by the number of bits used to represent the mantissa. The more bits in the mantissa, the greater the precision."), [0003] ("While some applications may require these types of precision, other applications may not. For example, some graphics applications may only require a 16-bit mantissa. For these graphics applications, any accuracy beyond 16 bits of precision tends to result in unnecessary power consumption. ... Accordingly, there is a need in the art for a *floating-point processor in which the reduced precision, or subprecision, of the floating-point format is selectable.*").

388. Dockser teaches an FPP in which the "precision" for "floating-point operations may be reduced" by "selecting [a] subprecision" to whatever precision level is "needed." Dockser, [0014] ("In at least one embodiment of a floating-

point processor, the precision for one or more floating-point operations may be reduced from that of the specified format. ... By selecting the subprecision of the floating-point format, to that needed for a particular operation, thereby reducing the power consumption of the floating-point processor to support the selected subprecision, greater efficiency as well as significant power savings can be achieved.”).

389. Dockser further teaches that the way it achieves a reduction in precision is by dropping mantissa bits. *See* Dockser, [0024] (“one or more computational units in the floating-point operator 140 may execute the instructions of the requested floating-point operation on the received operands, at the subprecision selected by the floating-point controller 130”), [0025] (“The subprecision select bits are written to the control register 137, which in turn controls the bit length of the mantissa for each operand during the floating-point operation.”), [0026] (“the floating-point controller 130 may cause power to be removed from the floating-point register elements for the excess bits of the fraction that are not required to meet the precision specified by the subprecision select bits”), [0027] (“[T]he logic in the floating-point operator 140 corresponding to the excess mantissa bits do not require power. Thus, power savings may be achieved by removing power to the logic in the floating-point operator 140 that remains unused as a result of the subprecision selected.”). Thus, Dockser teaches that the

number of fraction bits dropped/maintained is a result-effective variable that affects the results of both precision and power.

H. Claim 18: “The device of claim 8, wherein the dynamic range of the possible valid inputs to the first operation is at least as wide as from 1/1,000,000 through 1,000,000.”

390. Claim 18 of the ’273 patent depends from claim 8 and changes the minimum dynamic range from what is recited in claim 1 to “at least as wide as from 1/1,000,000 through 1,000,000.”

391. As I explained in Section VI.B.3 above when discussing limitation [1B1], the dynamic range of possible valid inputs to Dockser’s operation extends from approximately 2^{-126} (much smaller than 1/1,000,000) to approximately 2^{127} (much larger than 1,000,000), meeting claim 18 of the ’273 patent.

I. Claim 19: “The device of claim 1, wherein the at least one first LPHDR execution unit comprises a plurality of locally connected LPHDR execution units.”

392. As I explained in Section VIII.B above, the Dockser/MacMillan combination includes multiple MacMillan PEs, each of which contains Dockser’s LPHDR execution unit. MacMillan’s SIMD-RAM includes “nearest neighbor interconnections... between PEs,” which a POSA would have understood are *local* connections that MacMillan describes as being different from “global interconnections.” MacMillan, 15:51-6:12 (“A serial communications scheme could be used to further reduce wiring complexities. *In addition to global*

interconnections provided by the Interconnect Router 256, *each PE could also contain logic for nearest neighbor interconnections*. This facilitates rapid up/down/left/right shifts of data *between PEs*, as is common, for example, in image and signal processing. The use of an Interconnect Router 256 provides high speed interprocessor communication. It is possible to use the described invention without an Interconnect Router 256 by using only the nearest neighbor interchip communications of the SIMD-RAM chip 254, and the data movement capabilities of the Host CPU 208 and SIMD Controller 252, although such an implementation may reduce the performance of interprocessor communications.”),

Figure 5.

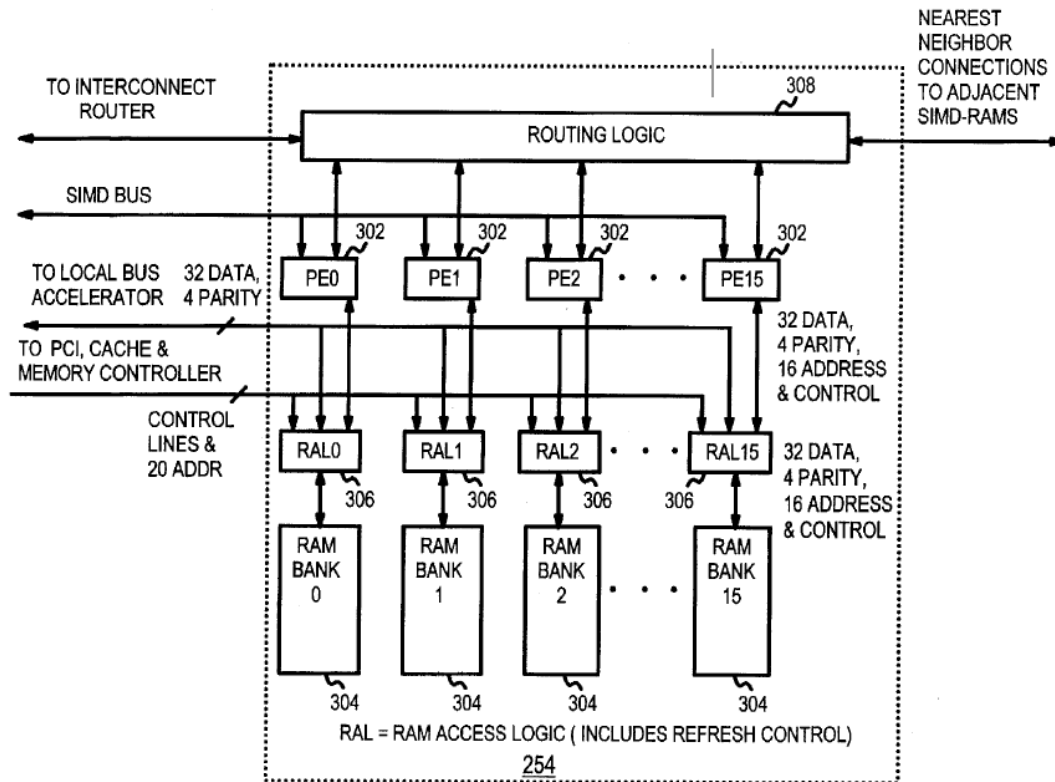


FIG. 5

393. The '273 patent similarly describes “local connections” as nearest-neighbor connections between PEs. '273 Patent, 9:37-51 (“The PEA 104 itself, besides communicating with the CU 106 and IOU 108 and possibly other mechanisms, has ways for data to move within the array. For example, the PEA 104 may be implemented such that data may move from PEs *only to their nearest neighbors, that is, there are no long distance transfers*. FIGS. 2 and 3 show embodiments of the present invention which use this approach, where the nearest neighbors are the four adjacent PEs toward the North, East, West, and South, called a NEWS design. ... For instance, every PE might access a specified data value in its neighbor to the West and copy it into its own local storage.”), 10:23-25 (“The local connections [between PEs] may include a mixture of various distance hops, such as ... distance 1,” *i.e.* nearest-neighbor), 16:36-38 (embodiment of invention “provides the arithmetic/memory units in a two-dimensional physical layout with only local connections between units”).

394. Dockser/MacMillan’s multiple LPHDR execution units thus “comprise[] a plurality of locally connected LPHDR execution units” (Dockser’s execution units connected via local PE connections), meeting claim 19 of the '273 patent.

J. Claim 20: “The device of claim 1, wherein the device has a SIMD architecture.”

395. In the Dockser/MacMillan combination, MacMillan’s computer system or SIMD-RAM device meets the ’273 patent’s “device of claim 1,” as I explained in Section VIII.C above. MacMillan’s computer system and SIMD-RAM device both “ha[ve] a SIMD architecture” as claim 20 of the ’273 patent recites. MacMillan, 2:13-14 (“The *SIMD architecture* can be scaled to tens of thousands of processors or more...”), 5:22-45 (“SIMD architectures have some qualities that make them attractive candidates for adding supercomputer performance to a computer for personal use. However, as a result of the limitations described above, present SIMD architectures are much higher cost than computer systems for personal use. ...Overcoming the above limitations would allow supercomputing performance to be provided in a low cost computer system for personal use, dramatically expanding the potential market for systems with supercomputer performance. ... The present invention addresses the above needs.”), 6:51-65 (“FIG. 2 is a block diagram of a first embodiment of a single instruction multiple data (*SIMD*) computer system ... FIG. 3 is a block diagram of a second embodiment of a single instruction multiple data (*SIMD*) computer system, in accordance with the present invention ... FIG. 4 is a block diagram showing the interaction of the SIMD Controller with the Host CPU ... in

accordance with the present invention. FIG. 5 is a block diagram of a *SIMD*-RAM device in accordance with the present invention.”).

K. Claim 21: “The device of claim 1, wherein the device includes memory locally accessible to the at least one first LPHDR execution unit.”

396. Dockser/MacMillan’s SIMD-RAM device includes memory (DRAM) coupled to multiple PEs on a SIMD-RAM chip, meeting claim 21’s “memory locally accessible to the... LPHDR execution unit” (Dockser’s execution unit in MacMillan’s PE):

FIG. 5 shows a block diagram of a SIMD-RAM device in accordance with the present invention. The SIMD-RAM device 254 shows a *plurality of memory devices, in this case, dynamic random access memory (DRAM) 304, which are coupled to a plurality of processing elements (PE's) 302 via random access logic 306.* ... The embodiment of FIGS. 2, 3, and 5 show the invention with dynamic random access memory (DRAM) technology used to implement the memory storage locations in the SIMD-RAM 254. It is within the spirit and scope of the invention to use other technologies to implement the memory storage locations.

MacMillan, 12:35-13:10; *see also* MacMillan, FIG. 5

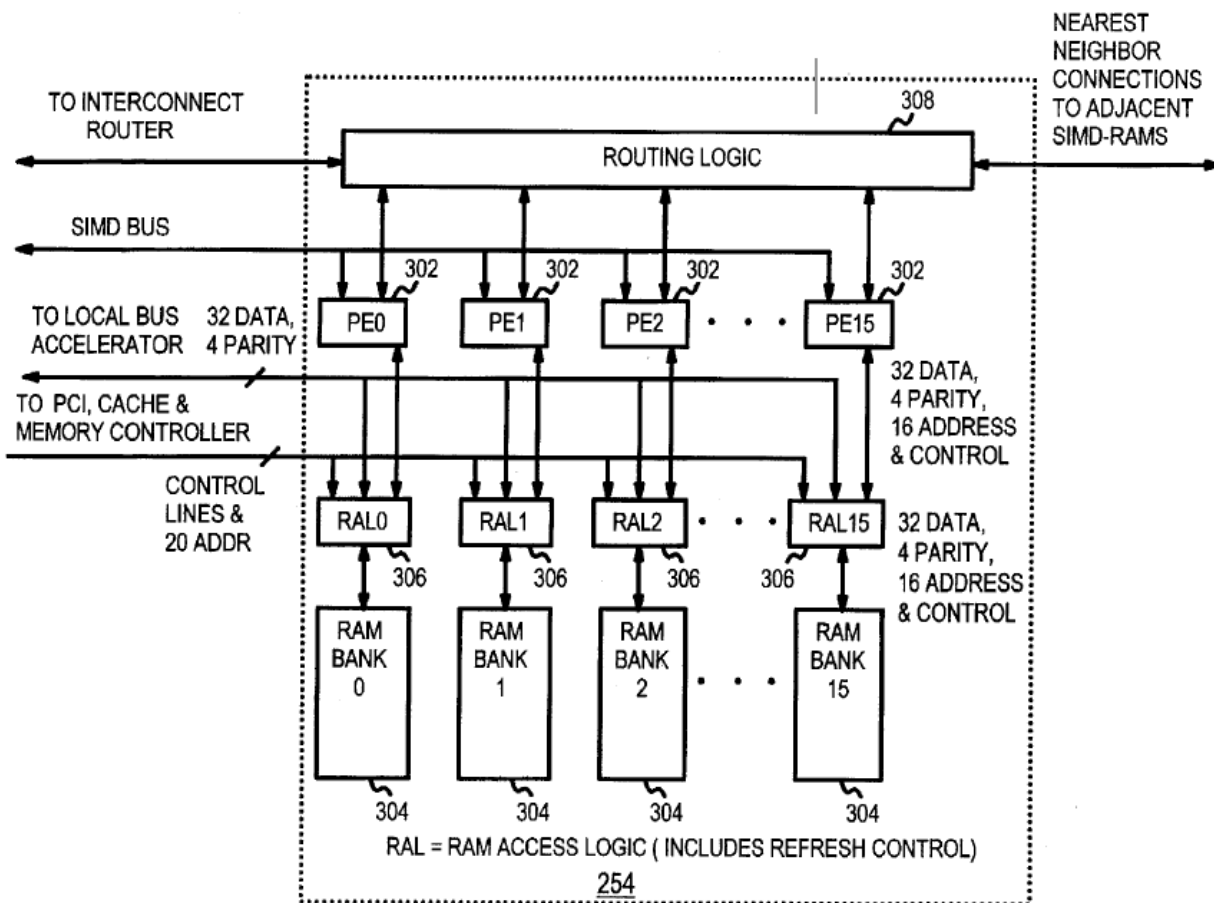


FIG. 5

397. Alternatively, the registers in Dockser's execution unit within the Dockser/MacMillan system also meet claim 21 of the '273 patent for the reasons I discussed in Section VI.D above.

L. Claims 22-23

398. Dockser/MacMillan's SIMD-RAM device is implemented on a "chip," which a POSA would have understood to be a silicon chip as claims 22-23 of the '273 patent recite. MacMillan, 12:60-13:4 ("The architecture of the SIMD-

RAM 254 allows scaling to higher or lower density *chips* with more or fewer PEs
 302 ... If the memory-per-PE ratio (64 kbytes-per-PE in the above example) is
 maintained, lower or higher density SIMD-DRAM 254 *chips* could be used ...

With this approach, an application program would see no difference, for example,
 between a system implemented with sixteen SIMD-RAM 254 *chips*, each having 1
 Mbyte DRAM and 16 SIMD PEs, and a system implemented with a single SIMD-
 RAM 254 *chip* containing 16 Mbytes of DRAM and 256 PEs.”).

399. Alternatively, to the extent MacMillan is not considered to disclose
 that the “chip” is silicon, that would have been the typical, conventional, and
 obvious implementation of MacMillan’s “chip.” For corroborating evidence, *see*,
e.g., Kacker (Ex. 1055), [0005] (“Integrated circuits (chips) are generally made of
 silicon on which electronic circuits are fabricated.”).

400. Claim 23 of the ’273 patent additionally recites that the device “is
 implemented... using digital technology.” A POSA would have understood that
 Dockser/MacMillan’s SIMD-RAM device is implemented using digital
 technology, as claim 23 recites, because, *e.g.*, the SIMD-RAM device operates on
 bits. MacMillan, 12:47-49 (“Each PE contains a 32-bit wide data path and can
 perform atomic operations on *bits, bytes, 16-bit words, and 32-bit words.*”), 13:20-
 14:22 (referring to sizes of “words” on which the PEs operate in terms of numbers

of “bits”). *See also* Section VI.F above (where I explained that devices that operate on “bits” are digital devices).

M. Claim 24: “The device of claim 1, wherein the device further comprises a digital processor adapted to control the operation of the at least one first LPHDR execution unit.”

401. MacMillan’s computer system in the Dockser/MacMillan combination includes a Host CPU implementable as “a 386, 486 or Pentium™ processor” (MacMillan, 9:30-31), each of which a POSA would have understood is a well-known “digital processor” as claim 24 of the ’273 patent recites. As I explain in paragraphs 402-405 below, a POSA would have understood that MacMillan’s Host CPU in the Dockser/MacMillan combination is adapted to control the operation of the SIMD subsystem, including its PEs with Dockser’s execution units, by executing a program that controls initiation of “all SIMD processing” (MacMillan, 10:24-53, 13:12-13, 13:38-62); thus meeting claim 24 of the ’273 patent.

402. MacMillan explains that in the embodiment shown in Figure 3, the “computer system” includes a “[h]ost CPU” and a “SIMD subsystem 250” with a “SIMD Controller.” MacMillan, 9:20-31. MacMillan explains that “SIMD programs” can be “linked to traditional Host CPU [] programs,” and that “[u]sing this convention, *all SIMD processing* would be *initiated from a uniprocessor program*.” MacMillan, 10:27-30. The “Host CPU” initiates SIMD processing by “call[ing] a system subroutine to start SIMD Controller execution,” MacMillan,

10:46-51 and the SIMD controller “executes the SIMD program,” MacMillan, 10:66-67.

403. A POSA would have understood that “uniprocessor program” refers to a program run on the “host CPU,” because MacMillan refers to the “conventional” system shown in Figure 1, which includes a “[h]ost CPU” but no SIMD component, as a “uniprocessor system” MacMillan, 8:56-9:2, and because MacMillan refers to a “situation where the host CPU 208 is executing a uniprocessor program,” MacMillan, 10:44-45. A POSA would have understood that because “the PEs operate under the control of the SIMD controller,” MacMillan, 13:35, and because a “uniprocessor program” running on the “host CPU” “start[s] SIMD controller execution,” MacMillan, 10:44-49, the “host CPU” is adapted to control the PEs. Because in the Dockser/MacMillan combination, the “floating point accelerators” on each PE are the execution units of Dockser, this means that the “host CPU” in MacMillan’s system is adapted to control the execution units. This scheme, with a host CPU controlling PEs through a controller, mirrors the control scheme described in the ’273 patent. *See* ’273 patent, 8:29-30 (“The Host 102 is responsible for overall control of the computing system 100.”), 8:48-49 (“the specialized control unit (CU) 106 may be included in the architecture”), 8:60-64 (“In a design which includes the CU 106, the Host 102 typically will load the instructions (the program) for the PEA 104 into the CU

instruction memory (not shown in FIG. 1), then instruct the CU 106 to interpret the program and cause the PEA 104 to compute according to the instructions.”).

404. MacMillan explains that “[t]he SIMD Controller can pass data values to and from the Host CPU through the shared SIMD-RAM memory.” MacMillan. 13:12-13. *See also* MacMillan, 13:60-62 (“A key issue in memory mapping is how data arrays and structures are passed between CPU programs and SIMD routines.”). A POSA would have understood that by passing data to “SIMD routines,” the Host CPU is passing data to the PEs, since the PEs will access that data as instructed by the “SIMD subroutines.” *See, e.g.*, 13:35-37 (“Since the PEs operate under the control of the SIMD Controller and do not directly execute instructions from memory, they can use memory only for holding data.”). A POSA would have understood that when the Host CPU sends data to the PEs, it is controlling the PEs, and therefore controlling the execution units on the PEs in the Dockser/MacMillan combination, by controlling the data the PEs operate on, similar to what is described in the ’273 patent. *See, e.g.*, ’273 patent, 8:37-40 (“A goal of the Host 102 is to have the PEA 104 perform massive amounts of computation in a useful way. It does this by causing the PEs to perform computations, typically on data stored locally in each PE...”), 9:6-9 (“In order to get data into and out of the CU 106 and PEA 104, the I/O Unit 108 may interface the CU 106 and PEA 104 with the Host 102, the Host’s memory (not shown in

FIG. 1), and the system's Peripherals 110...”), 18:5-17 (describing “an algorithm which may be performed by machines implemented according to embodiments of the present invention” in which “[t]he Examples are placed into the memories associated with the PEs...[and] [g]iven a Test, the Test is passed through all the PEs, in turn”).

405. Thus, because in Dockser/MacMillan the “Host CPU” (8:56-9:31) in MacMillan’s system controls the LPHDR execution units (Dockser’s execution units on each PE), the “Host CPU” meets the “digital processor adapted to control the operation of the at least one first LPHDR execution unit” recited in claim 24 of the ’273 patent.

N. Claim 25

406. Claim 25 of the ’273 patent combines limitations of claims 9, 19, and 21-23, as I discussed in paragraph 171 (Section V.D.13) above.

Dockser/MacMillan’s SIMD-RAM device meets the limitations in claims 19 and 21-23 as I discussed in Sections VIII.I and VIII.K-VIII.L above. The SIMD-RAM can be “scal[ed] to higher... density... with more... PEs” than the “example” SIMD-RAM chip with “256 PEs” (MacMillan, 12:60-13:4); thus a SIMD-RAM with 500 PEs meeting the limitation in claim 9 would have been obvious. (*See my related discussion of claim 9 in Section VIII.D above.*) This obvious implementation of Dockser/MacMillan’s SIMD-RAM device meets all the

limitations as arranged in claim 25, and therefore meets claim 25 of the '273 patent.

O. Claim 26: “The device of claim 1, wherein the device is part of a mobile device.”

407. Dockser/MacMillan’s computer system and SIMD-RAM (*see* Section VIII.B above), either of which meets the “device of claim 1” as I explained in Section VIII.C above, are “part of a mobile device” as claim 26 of the '273 patent recites—*e.g.*, a “palmtop[], personal digital assistant[],...[etc.].” MacMillan, 7:16-34 (“‘computer system for personal use’ includes ‘palmtops, personal digital assistants, notebooks, subnotebooks, and video games’”).

P. Claims 36-61 and 63

408. Limitations [36A1]-[36B2] of claim 36 of the '273 patent are identical to limitations [1A1]-[1B2] of claim 1, and are met in the Dockser/MacMillan combination for the reasons I discussed in Section VI.B above regarding claim 1. Limitation [36C] is identical to claim 5, except that limitation [36C] is broader by omitting the “by at least ten” that is recited in claim 5. Dockser/MacMillan thus meets limitation [36C] for the same reasons I discussed in Section VIII.E above regarding claim 5, and Dockser/MacMillan renders obvious claim 36.

409. Claims 37-61 and 63 of the '273 patent depend from claim 36, but otherwise are identical to claims 2-26 and 28, respectively, which depend from claim 1. The limitations recited in claims 37-61 and 63 thus are met by

Dockser/MacMillan for the reasons I discussed in Sections VIII.C-VIII.O above with respect to claims 2-26 and 28.

IX. CLAIMS 1-26, 28, 32-61, 63, AND 67-70 WOULD HAVE BEEN OBVIOUS OVER DOCKSER, TONG, AND MACMILLAN

A. Claims 1-26, 28, 32, 36-61, 63, and 67

410. As I discussed in Section VIII.B above, Dockser/MacMillan uses Dockser's FPP to implement each "floating-point accelerator" in the parallel PEs of MacMillan's SIMD architecture. In addition, as I discussed in Section VII above, a POSA would have been motivated to implement Dockser's FPP using precision levels as low as 5 mantissa fraction bits, which Tong teaches suffices for applications including the "neural network trainer" "ALVINN." Tong, p. 278 & Table IV; *see* Sections VII.A-VII.B above.

411. MacMillan teaches that such "neural net[work]" applications would benefit from "supercomputer performance" provided by using (at least) 256 PEs including "floating point accelerators." MacMillan, 1:24-37, 12:52-13:4.

MacMillan states:

A personal computer with supercomputer performance could stimulate development of new software applications. There are many *applications that could benefit from supercomputer performance in a personal computer. Examples* of these applications are signal processing, data compression and decompression, phased-array radars, cryptography, pattern recognition, optimization, genetic

algorithms, **neural nets**, decision support, database mining, statistical analysis, text retrieval and categorization, simulation and modeling, medical imaging, medical signal processing, data visualization, optical character recognition, audio processing, voice synthesis and recognition, speech analysis, vision systems, video processing, multimedia, virtual reality, and CAD/CAM.

MacMillan, 1:24-37. MacMillan then explains that in its SIMD system, “floating point accelerators could be included in each PE” in the SIMD-RAMs. MacMillan, 12:52-59. MacMillan also explains that “application programs” can take advantage of its SIMD-RAM architecture:

The **architecture of the SIMD-RAM 254** allows scaling to higher or lower density chips with more or fewer PEs 302, more or less memory 304, and different amounts of memory per PE. If the memory-per-PE ratio (64 kbytes-per-PE in the above example) is maintained, lower or higher density SIMD-DRAM 254 chips could be used without software changes. **With this approach, an application program** would see no difference, for example, between a **system implemented with sixteen SIMD-RAM 254 chips**, each having 1 Mbyte DRAM and 16 SIMD PEs, and a **system implemented with a single SIMD-RAM 254 chip** containing 16 Mbytes of DRAM and 256 PEs.

MacMillan, 12:60-13:4.

412. In view of these combined teachings of Dockser, Tong, and MacMillan that I discuss in the two preceeding paragraphs, a POSA would have

been motivated to use Dockser's FPP with Tong's precision levels as low as 5 mantissa fraction bits as each floating-point accelerator in MacMillan's multiple PEs to achieve "supercomputer performance" (MacMillan, 1:24) for, *e.g.*, neural network applications while conserving power.

413. The resulting Dockser/Tong/MacMillan combination (*i.e.*, MacMillan's multi-PE system using Dockser's FPP with precision levels as low as 5 fraction bits as taught by Tong to implement each of MacMillan's floating-point accelerators) meets claims 1-26, 28, 36-61, and 63 of the '273 patent for the same reasons that Dockser/MacMillan does, which I discussed in Section VIII above, except that the claimed performance characteristics X and Y are met by Dockser's FPP using 5 rather than 9 mantissa fraction bits, meeting the claimed "at least" X and Y values by the even greater amounts that I explained in connection with Dockser/Tong in Section VII.B above.

414. Dockser/Tong/MacMillan meets claims 32 and 67 of the '273 patent because a POSA would have had reason to implement the Dockser/Tong/MacMillan device to "perform nearest neighbor search" for the reasons I discussed in Section VII.C above. Both MacMillan and Tong teach applying their techniques in applications including neural network and signal/image processing applications, for which nearest-neighbor search was known to be advantageous. Tong, p. 278 & Table IV; MacMillan, 1:24-37.

B. Claims 33-35 and 68-70

415. Tong’s teachings to emulate in software a device employing reduced-precision arithmetic (which I discussed in Sections VII.A, VII.D above) would have motivated a POSA to emulate in software the Dockser/Tong/MacMillan device that meets claim 1 of the ’273 patent (which is identical to the “second device” recited in claim 33) and claim 36 (which is identical to the “second device” recited in claim 68), since the Dockser/Tong/MacMillan device also employs reduced-precision arithmetic, as I explained in Section IX.A above. This emulated Dockser/Tong/MacMillan device meets the “second device” recited in claim 33 and the “second device” recited in claim 68.

416. The computer performing the emulation meets claim 33 of the ’273 patent for the same reasons that computer meets claim 33 in the Dockser/Tong combination, which I discussed in Section VII.D above, except that the claimed “second device” being emulated is met by the Dockser/Tong/MacMillan device for the reasons I discussed in Section IX.A above. The emulation computer also meets claim 68’s preamble in the same way as it meets claim 33’s identical preamble (as I discussed in Section VII.D above), and claim 68’s remaining limitations (concerning the emulated “second device”) are met for the same reasons the emulated Dockser/Tong/MacMillan device meets claim 36’s identical limitations (as I discussed in Section VIII.P above).

417. The Dockser/Tong/MacMillan device being emulated meets the additional limitations of claims 34-35 and 69-70 of the '273 patent for the same reasons I discussed in Sections VIII.D-VIII.E above concerning claim 3 (which recites identical limitations to claims 34 and 69) and claim 5 (which recites identical limitations to claims 35 and 70).

C. Alternative Interpretation of Claims 5-6, 8, 10-18, and 35-70

418. One embodiment in MacMillan is “a system... designed primarily... for a specific application, including embedded applications... including... signal processing,... voice recognition, [etc.]” MacMillan, 7:15-34. MacMillan relates to improving a “computer system for personal use.” MacMillan, 5:48-53 (“The present invention comprises a computer system to which has been added a Single Instruction Multiple Data (SIMD) parallel processing capability. The computer system, to which the SIMD capability has been added, could be a computer system for personal use or other computer system.”). And in the “DESCRIPTION OF THE PREFERRED EMBODIMENTS” section, MacMillan explains that:

‘A computer system for personal use’ is a system that is designed primarily for use by an individual or for a specific application, including embedded applications. A computer system for personal use includes personal computers (including, but not limited to, those that are IBM-compatible and Apple-compatible), workstations (including, but not limited to, those that use RISC technology or that use the UNIX operating system), embedded computers (including,

but not limited to, computer-based equipment used for

telecommunications, data communications, industrial control, process control, printing, settop boxes, *signal processing*, data compression or decompression, data transformation, data aggregation, data extrapolation, data deduction, data induction, video or audio editing or special effects, instrumentation, data collection or analysis or display, display terminals or screens, *voice recognition, voice processing, voice synthesis, voice reproduction*, data recording and playback, music synthesis, animation, or rendering), laptops, palmtops, personal digital assistants, notebooks, subnotebooks, and video games.

MacMillan, 7:15-34.

419. Tong says that its “results show that programs such as *speech recognition and image processing* use significantly less power with [a] reduced bitwidth FP representation than with an IEEE-standard FP representation,” and notes that “[i]t has long been known that many *such signal processing applications* can get by with less precision/range than full FP.” Tong, 273. Later, Tong explains:

For an increasing number of *embedded applications such as voice recognition, vision/image processing, and other human-sensory-oriented signal-processing applications*, FP’s simplified programming model (in contrast with fixed-point systems) and large dynamic range makes FP hardware a desirable feature. Furthermore, many recognition applications achieve a high degree of accuracy starting from fairly low-resolution input sensory data.

Tong, 274. Thus, a POSA would have understood that Tong teaches that many “embedded applications such as voice recognition,... and other human-sensory-oriented signal-processing applications” (Tong, 274), “can get by with less precision/range than full FP” (Tong, 273).

420. Tong experimentally demonstrates that the optimum precision for a “benchmark suite” (Tong, 284) of “five signal processing applications” spanning a “range in complexity” is between 5 and 11 mantissa fraction bits (Tong, 278-279). Tong says that “[t]o study the relationship between program accuracy and number of bits in FP representation, [the authors] have collected a set of five signal processing applications.” Tong, 278. These applications, which are described in Table IV, “range in complexity from single-purpose kernels to complete applications.” Tong, 278. Figure 6 of Tong “plots the accuracy for each of the five programs across a range of mantissa bitwidths.” Tong, 278. As shown in Figure 6, “[n]one of the workloads display a noticeable degradation in accuracy when the mantissa bitwidth is reduced from 23 to 11 bits. For ALVINN and Sphinx III the results are even more promising; the accuracy does not change significantly with as few as 5 mantissa bits.” Tong, 278. Later, in summarizing its results, Tong notes that “[m]ost FP programs in our benchmark suite maintain the same output even when the mantissa bitwidth is reduced by half.” Tong, 284.

421. A POSA would have been motivated to use Dockser's FPPs in MacMillan's architecture with Tong's precision levels for the reasons I explained in Section IX.A above. Moreover, a POSA would have been motivated to do so in MacMillan's embedded signal-processing system based on Tong's teachings that reduced-precision arithmetic is beneficial in such systems.

422. In such an embedded system designed specifically for signal processing, a POSA would have been motivated to customize Dockser's FPPs in MacMillan's PEs to only operate at precision levels lower than full FP 32-bit operations, in view of Tong's teachings that "the fine precision of the 23-bit mantissa is not essential." Tong, 279. Tong explains that the results of its experiments with varying mantissa bitwidths "clearly demonstrate that not all programs need the precision provided by generic FP hardware." Tong, 278. As Tong explains,

The reason behind this result is that *many programs dealing with human interfaces process sensory data with intrinsically low resolutions*. Raw input data with 4-10 bits of precision is rather common in these applications. ... What is quite clear from these experiments is that the FP format provides essential dynamic range (we can reduce, but not reduce dramatically, the number of exponent bits) *but the fine precision of the 23-bit mantissa is not essential* (half as many bits often suffice).

Tong, 278-279. Based on this teaching, a POSA would have understood that there is a class of applications, including signal processing applications that “process sensory data,” that do not need 23 fraction bits.

423. In addition, Dockser teaches that the registers in embodiments of its FPP can be “formatted differently from IEEE 32-bit single format.” Dockser, [0017] (“It should of course be understood, however, that other embodiments of the floating-point processor 100 may include a floating-point register file 110 that is *formatted differently* from IEEE 32-bit single format (*including but not limited to* IEEE 64-bit double format), and/or may contain a different number of register locations.” Thus, a POSA would have been motivated to implement Dockser’s FPPs in the embedded signal-processing system with smaller than 32-bit registers to not waste circuit space or incur unnecessary cost in having some register elements that will always be unpowered because they correspond to mantissa bits that will always be tied to “0” in an application-specific system that always operates at reduced precision.

424. Likewise, a POSA would have been motivated to implement the multiplier logic in Dockser’s FPP to have only as many logic elements as needed to multiply mantissas of the reduced bitwidth (smaller than 23-bit) corresponding to the precision level selected for the embedded application.

425. In this implementation, the claimed “input signal” to Dockser’s FPP remains in IEEE-754 32-bit single-format given that the Host CPU and data buses in the Dockser/Tong/MacMillan device use and send to Dockser’s FPP standard 32-bit floating-point numbers. MacMillan, 9:30-57 (“In this embodiment of FIG. 3 ... The host CPU 208 is connected to the host bus 230. In the embodiment of FIG. 3, fast I/O is provided through the Peripheral Component interconnect bus (PCI bus) 210. ... The PCI bus 210 can support up to 133 Mbytes/second peak transfer rate in its current 32-bit version.”); *see* Section IX.A above (where I explain that the device in the Dockser/Tong/MacMillan combination is the computer system of MacMillan implemented with Dockser’s FPPs, which are configured to use 5 mantissa bits as taught in Tong), VIII.A, VIII.E. However, in this implementation, one or more least-significant mantissa bits of the input signal number are not stored in the FPP register because it is implemented with fewer than 32 storage elements, as I explain in paragraph 423 above.

426. Because the input signal and selected precision levels are unchanged from the Dockser, Dockser/Tong, and Dockser/MacMillan combinations that I discuss in Sections VI-VIII above, this implementation of Dockser/Tong/MacMillan in paragraphs 422-425 above meets the independent claims of the ’273 patent (including their “wherein” clause) the same way as in the

Dockser, Dockser/Tong, and Dockser/MacMillan combinations where Dockser's FPP is a claimed LPHDR execution unit.

427. As I discuss in Section VIII.E above, the specification of the '273 patent makes clear to a POSA that the claimed "execution units... adapted to execute... multiplication on floating point numbers that are at least 32 bits wide" referred to in claims 5, 8, 10, 35-36, 40, 43, 45, 68, and 70 are "traditional precision" execution units that ***do not*** "sometimes" produce results different from the correct traditional-precision result. *See, e.g.*, paragraphs 371-372 above. However, if the above-listed claims (and their dependents) were interpreted differently than the specification discusses, such that an execution unit used for reduced-precision operations could also be a claimed "execution unit[]... adapted to execute... multiplication on floating point numbers... 32 bits wide" if its hardware were capable of 32-bit multiplication in some configurations (*e.g.*, if power were applied to all register and logic elements), then Dockser's FPP in the above-discussed Dockser/Tong/MacMillan implementation for an embedded application is ***not*** that type of execution unit, because its hardware (having smaller-than-32-bit registers and multiplier logic) is ***not*** capable of 32-bit multiplication.

428. Thus, this Dockser/Tong/MacMillan implementation meets that alternative interpretation of claims 5-6, 8, 10-18, and 35-70 of the '273 patent because only the Host CPU floating-point unit is a claimed "execution unit[]...

adapted to execute... multiplication on floating point numbers... 32 bits wide,” and the device includes the claims’ recited number more of Dockser’s FPP (LPHDR execution unit), as I explained in Sections VIII.E above.

X. APPENDIX I – TECHNICAL DETAILS OF RELATIVE ERROR ANALYSES

A. Dockser’s Multiplier Relative Error Is Independent of Exponent and Sign

429. In this section, I explain mathematically why it is the case that the relative error produced by performing floating-point multiplication on two floating-point numbers using Dockser’s precision-reduction techniques is independent of the exponent values and sign bits of the two floating-point numbers.

430. A POSA would have understood that if a floating-point number A has a sign bit S_A , a mantissa value M_A , and an exponent E_A , then the value of A is

$$A = (-1)^{(S_A)} \times 2^{(E_A)} \times (M_A)$$

See, e.g., Dockser, [0001] (“A floating-point representation of a number commonly includes a sign component, an exponent, and a mantissa. To find the value of a floating-point number, the mantissa is multiplied by a base (commonly 2 in computers) raised to the power of the exponent. The sign is applied to the resultant value.”). A POSA would have understood that positive numbers are represented by having a sign bit with value 0, and negative numbers are represented with a sign

bit with value 1. This is because, if $S_A = 0$, it means that the term $(-1)^{(S_A)}$ in the formula above has a value of 1, because $(-1)^0 = 1$. This in turn means that A is a positive number, because the other two terms are also positive numbers. On the other hand, if $S_A = 1$, it means that the term $(-1)^{(S_A)}$ in the formula above has a value of -1, because $(-1)^1 = -1$. This in turn means that A is a negative number

431. From basic mathematics that a POSA would have been familiar with, a POSA would have understood that given an input pair of floating-point numbers A and B with sign bits S_A and S_B , exponents E_A and E_B , and mantissas M_A and M_B , the exact mathematical calculation of the product $Q = A \times B$ is:

$$Q = (-1)^{(S_A \otimes S_B)} \times 2^{(E_A + E_B)} \times (M_A \times M_B)$$

where ‘ \otimes ’ denotes an XOR operation, which is a well-known binary mathematical operation whose function I explain below. As can be seen from the expression above, multiplying two floating-point numbers involves multiplying their mantissas and adding their exponents, as I explained in paragraph 271 above.

432. The term “XOR” is a shorthand for “exclusive OR,” which is a binary operation that takes two bits, each of which can be either a 0 or a 1, and produces a 1 as an output if **only** one of the inputs is a 1 (which is why it is called an “exclusive” or operation), and a 0 as an output otherwise. Computing the XOR of the two sign bits ensures that the output of the multiplication has the mathematically correct sign. If the two inputs A and B are both positive numbers,

their sign bits S_A and S_B are both 0, meaning that $S_A \otimes S_B = 0$, which in turn means that the term $(-1)^{(S_A \otimes S_B)}$ in the equation above has a value of 1 (because $-1^0 = 1$), which means that Q will be a positive number. If the two inputs A and B are negative, their sign bits S_A and S_B are both 1, meaning that $S_A \otimes S_B = 0$, which in turn means that the term $(-1)^{(S_A \otimes S_B)}$ in the equation above has a value of 1, which means that Q will be a positive number. But if *only one* of A and B is negative, then one sign bit will be 1 and the other will be 0, meaning that $S_A \otimes S_B = 1$, which in turn means that the term $(-1)^{(S_A \otimes S_B)}$ in the equation above has a value of -1, which means that Q will be a negative number.

433. Representing the mantissa product ($M_A \times M_B$) as V yields:

$$Q = (-1)^{(S_A \otimes S_B)} \times 2^{(E_A + E_B)} \times V$$

434. As I explain in Section VI.B.4 above when discussion limitation [1B2], when used in the calculation of the product of two floating-point numbers, both Dockser's register bit-dropping and logic bit-dropping techniques alter the product's mantissa but not its sign or exponent.

435. Letting V' be the altered mantissa product produced by Dockser's FPP using either technique, the product Q' of Dockser's reduced-precision multiplication is:

$$Q' = (-1)^{(S_A \otimes S_B)} \times 2^{(E_A + E_B)} \times V'$$

436. In the claims of the '273 patent, the variable Y is a relative error percentage. *See, e.g.*, paragraph 38 above. This claimed relative error percentage Y is calculated using the following formula:

$$Y = \left| \frac{Q - Q'}{Q} \right| \times 100$$

See, e.g., '273 patent, 26:55-27:4 (“For example, in certain embodiments, a LPHDR arithmetic element produces results which are sometimes (or all of the time) no closer than 0.05% to the correct result (that is, ***the absolute value of the difference between the produced result and the correct result*** is no more than one-twentieth of one ***percent of the absolute value of the correct result***). ... As another example, a LPHDR arithmetic element may produce results which are sometimes (or all of the time) no closer than 0.2% to the correct result. As yet another example, a LPHDR arithmetic element may produce results which are sometimes (or all of the time) no closer than 0.5%...[or] 1%, or 2%, or 5%, or 20% to the correct result.”).

437. Substituting the above expressions for Q and Q' into the equation for Y yields:

$$Y = \left| \frac{((-1)^{(S_A \otimes S_B)} \times 2^{(E_A + E_B)} \times V) - ((-1)^{(S_A \otimes S_B)} \times 2^{(E_A + E_B)} \times V')}{(-1)^{(S_A \otimes S_B)} \times 2^{(E_A + E_B)} \times V} \right| \times 100$$

438. Factoring out the exponent and sign terms yields:

$$Y = \left| \frac{((-1)^{(S_A \otimes S_B)} \times 2^{(E_A + E_B)}) \times (V - V')}{((-1)^{(S_A \otimes S_B)} \times 2^{(E_A + E_B)}) \times V} \right| \times 100$$

439. The exponent and sign terms in the numerator and denominator cancel out, yielding:

$$Y = \left| \frac{V - V'}{V} \right| \times 100$$

440. Thus, the claimed “Y” percentage produced by Dockser’s reduced-precision multiplication depends only on the operand mantissas and is independent of the exponents and signs.

B. Software Demonstration of Dockser’s Register Bit-Dropping

441. As I discussed in Section VI.B.4.c(1)(i) above, I wrote a software program (which a POSA would have understood how to write), that performs floating-point multiplication operations the way Dockser’s register bit-dropping technique does when dropping the 14 least-significant mantissa bits to zero as Dockser teaches. My program iterates through all possible pairs of normal IEEE-754 single-format floating-point numbers A and B that are non-negative (having ‘0’ sign bit) with zero-valued exponent (such that the mantissa is multiplied by $2^0 = 1$). As a POSA would have understood, the effect of doing this is to iterate over all possible pairs of normal IEEE-754 single-format mantissa values. My

program iterates over all possible pairs of normal IEEE-754 single-format mantissa values because in normal IEEE-754 single-format numbers, all of the possible normal mantissa values can be used with a 0 sign bit and a zero-valued exponent.

442. For each pair of numbers A and B , my program performs the following steps:

443. **Step 1:** Perform the exact mathematical calculation of product $Q = A \times B$.

444. **Step 2:** Retain a specified number of mantissa fraction bits for A and B and zero the remaining less-significant fraction bits to create reduced-mantissa operands A' and B' .

445. **Step 3:** Compute the product $Q' = A' \times B'$.

446. **Step 4:** Compute the relative error percentage $Y = \left| \frac{Q-Q'}{Q} \right| \times 100$; and

447. **Step 5:** If Y is at least the minimum percentage recited in a challenged claim, increment a counter. My program thus counts the number of input pairs for which Y is at least the minimum percentage recited in a challenged claim of the '273 patent.

448. As a POSA would have understood, testing only non-negative single-format floating-point numbers with zero-valued exponent is sufficient to test all possible pairs of single-format operands. This is because, as I explained in Appendix I.A above, the relative error percentage produced by Dockser's

multiplication is independent of operand exponent and sign, and because the set of all possible single-format operands combines all possible mantissas with all possible exponents and signs. This means that for all pairs of floating-point numbers A and B where A has the mantissa value M_A and B has the mantissa value M_B , the relative error Y that Dockser's FPP produces when multiplying A and B will be the *same, regardless* of the exponent and sign values of A and B. As I explain in Appendix I.E. below, after my program tested all possible mantissa pairs and returned a result, I adjusted the result by a multiplier that conservatively excludes pairs of floating-point numbers that could result in overflow/underflow exceptions.

449. After my program iterates through all possible pairs, it calculates the percentage of input pairs for which Y is at least the minimum percentage recited in the challenged claim being tested, by dividing the number in the counter by the total number of pairs that were tested, and then multiplying that quotient by 100 to yield a percentage. In other words, if T is the total number of pairs and N is the number of pairs for which Y is at least the minimum percentage being tested, my program computes:

$$\left(\frac{N}{T}\right) \times 100$$

450. My program counts pairs whose operands are the same numbers but in the opposite order (*e.g.*, the pairs (A=2, B=3) and (A=3, B=2)) as different "pairs."

A POSA would have understood that the percentage $(\frac{N}{T}) \times 100$ will be the same regardless of whether such pairs are counted as two different pairs or as a single pair, because counting them as a single pair would divide both N and T by two, resulting in the same percentage.

451. I wrote my program using the C programming language, which a POSA would have been familiar with or would have been able to learn. Furthermore, a POSA would have understood that an equivalent program could have been written using any of various other well-known programming languages, such as Perl. To reduce the run-time of my program, I ran my program on an NVidia Quadro GV100 GPU, using Nvidia's Compute Unified Device Architecture (CUDA) platform to parallelize my program by having different processing cores on the GPU test different subsets of the possible mantissa pairs in parallel. This too would have been within the level of ordinary skill of the POSA. In my program, both the number of retained fraction bits in the operands (corresponding to Dockser's selected precision level) and the Y percentage being tested are defined in the source code. To test different numbers of retained fraction bits and different minimum Y percentages, I changed those values and then re-compiled and re-ran my program. An exemplary version of the source code for my program testing $Y=0.05\%$ with 9 retained fraction bits is attached to my declaration as Attachment A1.

452. The results from my program shown in the table below demonstrate that Dockser’s register bit-dropping technique produces the claimed minimum relative error percentage Y for more than the claimed minimum percentage X of the possible valid inputs at precision levels retaining 9 fraction bits (Dockser, [0026]) or 5 fraction bits (Tong, 282) in the operands, for each claimed X/Y combination. For the reasons I explain in Appendix I.E. below, my analysis conservatively excludes pairs of operands that could result in overflow/underflow exceptions. Therefore, in the table below, I report not only the percentage of tested pairs that meet the claimed Y percentage as output by my software program, but also a “Dockser’s X” percentage that is conservatively adjusted to exclude possible overflow/underflow pairs, by multiplying the percentage output by my software program by an adjustment ratio, whose value I derive in Appendix I.E below. The “Dockser’s X” percentages, as the table below shows, are greater than the minimum X percentage recited in any of the ’273 patent’s claims, thus meeting the claims.

Retained Fraction Bits	Claimed Y%	Percentage of Tested Pairs Meeting Claimed Y%	Adjusted “Dockser’s X%”
9	≥ 0.05%	92.628313%	≥ 92.14333%
	≥ 0.10%	70.494075%	≥ 70.12498%
	≥ 0.15%	39.118049%	≥ 38.91323%
	≥ 0.20%	14.674747%	≥ 14.59791%

5	$\geq 0.05\%$	99.971764%	$\geq 99.44834\%$
	$\geq 0.10\%$	99.886981%	$\geq 99.36400\%$
	$\geq 0.15\%$	99.976929%	$\geq 99.22327\%$
	$\geq 0.20\%$	99.547206%	$\geq 99.02600\%$

453. A POSA would have understood that the claimed “exact mathematical calculation” of product $Q = A \times B$ can have 48 bits in its mantissa (the product of two 24-bit mantissas). This is because, as a POSA would have understood from basic mathematics, the product of two K -bit numbers can produce a result up to $2K$ bits wide. *See* Dockser, [0034] (“The output value, resulting from the floating-point multiplication described above, has a width (*i.e.* number of bits) that is equal to the sum of the widths of the two input values 402 and 404 that are being multiplied together.”), Fig. 3B (showing that multiplying a “k-BIT MULTIPLICAND” by a “k-BIT MULTIPLIER” produces a “2k-BIT OUTPUT VALUE”).

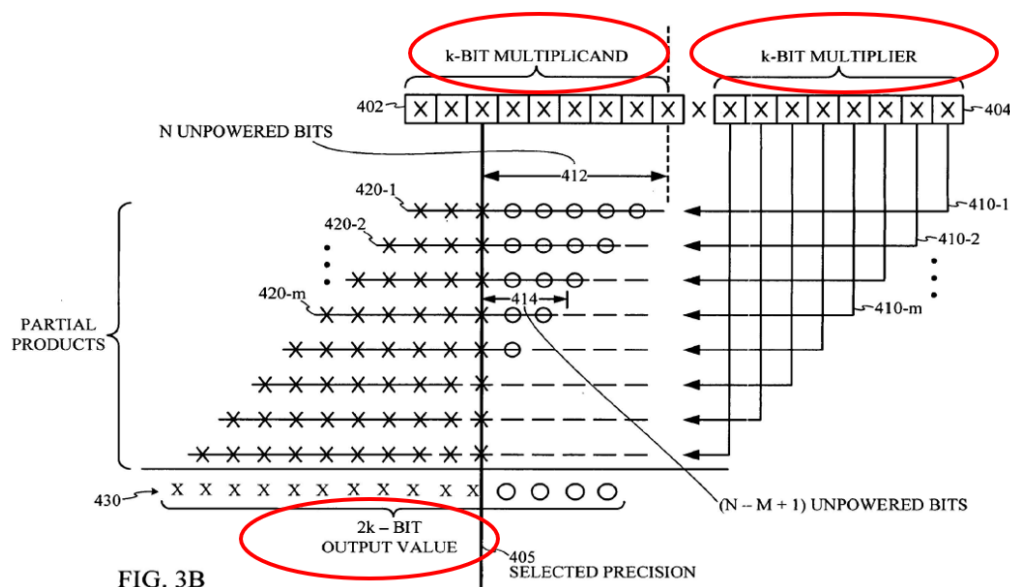


FIG. 3B

454. My program therefore stores the exact product in IEEE-754 double-precision floating-point format. This is because the double-precision format has space for up to 53 mantissa bits (52 stored fraction bits and a hidden bit) and is therefore wide enough to store all 48 mantissa product bits. *See* Dockser, [0002] (“The IEEE-754 also specifies a 64-bit double format having a 1-bit sign, an 11-bit exponent, and a 53-bit mantissa. Analogous to the single encoding, only the 52 fraction bits of the mantissa are stored in the 64-bit encoding, an integer bit, immediately to the left of the binary point, is implied.”).

455. If Singular were to argue (incorrectly) that the “exact mathematical calculation” of multiplication of two single-format floating-point operands is the product rounded or truncated to single format (having a 23-bit-fraction mantissa), the claims are still met. To confirm this, I wrote an alternative version of my software program for register bit-dropping (attached to this declaration as Attachment A2) that performed the same steps I discussed in paragraphs 442-447 above, except that it stores the product $Q = A \times B$ as a single-format (single-precision, having only up to 23 mantissa fraction bits) number. The table below reports the same percentages as the table I explained in paragraph 452 above, but using the alternate version of my program discussed in this paragraph. As the table below shows, my test using this alternative produced identical results up to at least

two decimal places of the “Dockser’s X” percentage as my test in paragraph 452 above that used the double-precision product as the exact mathematical calculation.

Retained Fraction Bits	Claimed Y%	Percentage of Tested Pairs Meeting Claimed Y%	Adjusted “Dockser’s X%”
9	$\geq 0.05\%$	92.628313%	$\geq 92.14333\%$
	$\geq 0.10\%$	70.494096%	$\geq 70.12500\%$
	$\geq 0.15\%$	39.118048%	$\geq 38.91323\%$
	$\geq 0.20\%$	14.674753%	$\geq 14.59792\%$
5	$\geq 0.05\%$	99.971764%	$\geq 99.44834\%$
	$\geq 0.10\%$	99.886980%	$\geq 99.36400\%$
	$\geq 0.15\%$	99.745509%	$\geq 99.22327\%$
	$\geq 0.20\%$	99.547206%	$\geq 99.02600\%$

C. Algebraic Analysis of Dockser’s Register Bit-Dropping

456. For single-format floating-point operands A and B with sign bits S_A and S_B , exponents E_A and E_B , and mantissas M_A and M_B , respectively, the exact mathematical calculation (V) of the product of multiplying the mantissas M_A and M_B is:

$$V = M_A \times M_B$$

Dockser’s register bit-dropping technique drops least-significant bits from (*i.e.*, truncates) the mantissa of each operand. *See* Section VI.B.4.c(1) above. Letting M'_A and M'_B be the truncated versions of mantissas M_A and M_B , and V' be the

product of M'_A and M'_B that Dockser's register bit-dropping technique produces, then:

$$V' = M'_A \times M'_B$$

457. As I demonstrated in Appendix I.A. (*see* paragraphs 434-439), the claimed relative error percentage “Y” is:

$$Y = \left| \frac{V - V'}{V} \right| \times 100$$

458. Letting D_A be the difference between M_A and M'_A , and letting D_B be the difference between M_B and M'_B , *i.e.*:

$$M'_A = M_A - D_A$$

$$M'_B = M_B - D_B$$

The output product of Dockser's register bit-dropping technique is then:

$$V' = M'_A \times M'_B = (M_A - D_A) \times (M_B - D_B)$$

459. Substituting the above expressions for V and V' into the expression for Y yields:

$$Y = \left| \frac{(M_A \times M_B) - ((M_A - D_A) \times (M_B - D_B))}{(M_A \times M_B)} \right| \times 100\%$$

460. Expanding the numerator to show all individual products yields:

$$Y = \left| \frac{((M_A \times M_B) - (M_A \times M_B) + (D_A \times M_B) + (M_A \times D_B) - (D_A \times D_B))}{(M_A \times M_B)} \right| \times 100\%$$

461. In the numerator, $(M_A \times M_B) - (M_A \times M_B)$ cancels out, yielding:

$$Y = \left| \frac{(D_A \times M_B) + (M_A \times D_B) - (D_A \times D_B)}{(M_A \times M_B)} \right| \times 100\%$$

462. The fraction can be re-written as the sum of three fractions:

$$Y = \left| \frac{D_A}{M_A} + \frac{D_B}{M_B} - \frac{(D_A \times D_B)}{(M_A \times M_B)} \right| \times 100\% \quad (\textbf{Equation A})$$

463. M_A and M_B are within the range $1 \leq \textit{mantissa} < 2$ because they are both normal mantissas. See my discussion of normal numbers in Section VI.A above (e.g., paragraph 197 above). Therefore $(M_A \times M_B) \geq M_B$, because the product of any two positive numbers greater than or equal to 1 is always greater than or equal to either of the individual numbers.

464. D_A and D_B are fractional differences between an original mantissa and its truncated version, because the truncation of the mantissas changes the value of the fractional part of the mantissa. Thus, D_A and D_B are both less than 1, and therefore $(D_A \times D_B) < D_B$, since the product of two positive numbers that are less than 1 is always smaller than either of the two individual numbers.

465. Therefore:

$$\frac{D_B}{M_B} \geq \frac{(D_A \times D_B)}{(M_A \times M_B)}$$

(because the right-hand fraction has a smaller numerator and a larger denominator than the left-hand fraction), and thus the quantity $\frac{D_B}{M_B} - \frac{(D_A \times D_B)}{(M_A \times M_B)}$ in Equation A is a positive number.

466. Thus, the relative error percentage is at least as large as the first fraction in Equation A; *i.e.*:

$$Y \geq \left| \frac{D_A}{M_A} \right| \times 100\% \quad (\text{Equation B})$$

The ratio $\frac{D_A}{M_A}$ expressed as a percentage therefore provides a lower bound on the relative error percentage resulting from Dockser's multiplication with register bit-dropping on any two operands.

467. From basic binary mathematics, a POSA would have understood that 50% of all possible values of M_A have a zero as the first (most-significant) fraction bit (*i.e.*, $M_A = 1.0\dots$); the other 50% have a one as the first bit (*i.e.*, $M_A = 1.1\dots$). This is because there are only two possible values for the most-significant fraction bit—0 and 1—and each can be combined with all possible combinations of the rest of the less-significant bits. Therefore out of the set of all possible normal mantissa fraction values, half of them will have 0 in the first bit, and half will have 1 in the first bit.

468. The value of D_A (the difference between the full and truncated mantissas of operand A) is determined by the $(K + 1)^{th}$ through the 23rd fraction

bits of M_A , where K is the number of fraction bits to which M_A is truncated in Dockser's register bit-dropping; *e.g.*, when $K=9$ (*see* Dockser, [0026]), M_A 's 10th through 23rd fraction bits are zeroed.

469. Of the 50% of M_A values that have a first fraction bit of zero, 25% have ones in both the $(K + 1)^{th}$ and $(K + 2)^{th}$ bits. This is because, as a POSA would have understood, there are only four possible values that pair of bits could have: 00, 01, 10, and 11, and each of those can be combined with all possible combinations of the other bits in the mantissa. The set of values that have a first fraction bit of zero and ones in both the $(K + 1)^{th}$ and $(K + 2)^{th}$ bits represents 12.5% of all possible values of M_A (*i.e.*, 25% of 50%).

470. In this 12.5% of all possible values of M_A , the value of D_A is at least $(2^{-(K+1)} + 2^{-(K+2)})$, which is the difference produced when ones in both the $(K + 1)^{th}$ and $(K + 2)^{th}$ bits are changed to zeros; and the value of M_A is no larger than $(1.5 - 2^{-23})$, which is the largest possible mantissa having a zero as the first fraction bit (*i.e.*, 1.0111...).

471. Referring to Equation B, therefore, the following inequality holds for the 12.5% of all possible M_A values with zero in the first fraction bit and ones in

the $(K + 1)^{th}$ and $(K + 2)^{th}$ fraction bits, when Dockser's register bit-dropping truncates the operands to K fraction bits:

$$Y \geq \frac{(2^{-(K+1)} + 2^{-(K+2)})}{(1.5 - 2^{-23})} \times 100\% \quad (\text{Equation C})$$

472. Since each M_A value can be paired with every possible value of M_B , the above Equation C is also true for 12.5% of all possible *pairs* of mantissas M_A and M_B . Thus, for a given subprecision retaining K fraction bits with Dockser's register bit-dropping technique, over 12% of all possible valid pairs of input operands will produce *at minimum* the relative error given by Equation C. (The percentage of pairs is 12.5% multiplied by the adjustment value I derive in Appendix I.E below to conservatively exclude pairs of floating point numbers that might produce overflow/underflow exceptions when multiplied, which is still a percentage larger than 12%).

473. Similarly, 25% of all possible values of M_A have zeros as the first *two* fraction bits (*i.e.*, $M_A = 1.00\dots$), and 25% of those have ones in both the $(K + 1)^{th}$ and $(K + 2)^{th}$ bits. In this 6.25% of all possible values of M_A (25% of 25%), the value of D_A is at least $(2^{-(K+1)} + 2^{-(K+2)})$, and the value of M_A is no larger than $(1.25 - 2^{-23})$, which is the largest possible mantissa having zeros as the first two fraction bits (*i.e.*, $1.00111\dots$). Therefore, the following inequality holds for the 6.25% of all possible input pairs in which M_A has zeros in the first two fraction bits

and ones in the $(K + 1)^{th}$ and $(K + 2)^{th}$ fraction bits, when Dockser's register bit-dropping truncates the operands to K fraction bits:

$$Y \geq \frac{(2^{-(K+1)} + 2^{-(K+2)})}{(1.25 - 2^{-23})} \times 100\% \quad (\text{Equation D})$$

474. Thus, for a given subprecision retaining K fraction bits with Dockser's register bit-dropping technique, over 6% of all possible valid pairs of input operands will produce *at minimum* the percent error given by Equation D. (The percentage of pairs is 6.5% multiplied by the adjustment value I derive in Appendix I.E below to conservatively exclude pairs of floating point numbers that might produce overflow/underflow exceptions when multiplied, which is still a percentage larger than 6%).

475. The table below provides the results of evaluating Equations C and D with various values K of retained fraction bits as the selected precision level.

Retained Fraction Bits (K)	Equation C: Minimum Y for $X \geq 12\%$	Equation D: Minimum Y for $X \geq 6\%$	Meets X/Y Percentages Recited by Claims:
9	$\geq 0.0976\%$	$\geq 0.1171\%$	1, 11-12, 33, 36, 46-47, 68
8	$\geq 0.1953\%$	$\geq 0.2343\%$	All above plus 13-16, 48-51
7	$\geq 0.3906\%$	$\geq 0.4687\%$	All above plus 17, 52
5	$\geq 1.5625\%$	$\geq 1.8750\%$	All above

D. Software Demonstration of Dockser's Logic Bit-Dropping

476. As I discussed in Section VI.B.4.c(2) above, I wrote a software program (which a POSA would have understood how to write) that performs

floating-point multiplication operations in the way Dockser's logic bit-dropping technique does by dropping to zero the mantissa partial product bits that are less significant than the 9-bit selected precision level of the output value.

477. My software program for Dockser's logic bit-dropping iterates through all possible pairs of non-negative ('0' sign bit) normal IEEE-754 single-format floating-point numbers A and B with zero-valued exponent (mantissa is multiplied by $2^0 = 1$), and for each pair performs the following steps.

478. **Step 1:** Perform the exact mathematical calculation of $Q = A \times B$;

479. **Step 2:** Perform Dockser's logic bit-dropping to compute Q' from A and B , in the manner I explain in paragraphs 483-487 below;

480. **Step 3:** Compute the relative error percentage difference $Y = \left| \frac{Q-Q'}{Q} \right| \times 100$; and

481. **Step 4:** If Y is at least the minimum percentage recited in a challenged claim, increment a counter. The program thus counts the number of input pairs for which Y is at least the minimum percentage recited in a challenged claim of the '273 patent.

482. Testing all possible pairs of non-negative single-format floating-point numbers with zero-valued exponent suffices for the reasons I discussed in Appendix I.B above (*see* paragraph 448) in connection with my analogous software program for register bit-dropping. After my program iterates through all

possible pairs, it calculates the percentage of input pairs for which Y is at least the minimum percentage recited in a challenged claim being tested, in the same manner I discussed in paragraphs 420-421 above with respect to my register bit-dropping program.

483. My program computes Q' (step 2, paragraph 479 above) using the following technique, which is the technique Dockser describes at [0031]-[0034] and Figure 3B:

484. **Step 1:** Initialize the “output value” of the mantissa product to 0. Dockser, [0031] (“floating-point multiplication is performed in a series of stages ... one partial product is generated for every bit in the multiplier 404 ... The partial products or shifted floating-point numbers 420-i are added together to generate the *output value* 430 for the multiplication”).

485. **Step 2:** Interpret the bit sequences of the mantissas of operands A and B (including the implied leftmost “1” bit) as 24-bit integers representing the “multiplicand 402” and “multiplier 404.” Dockser, [0030] (“Binary multiplication as illustrated in FIG. 3B is basically a series of additions of shifted floating-point numbers. In the illustrated embodiment, binary multiplication is performed between a k-bit multiplicand 402 and a k-bit multiplier 404 ...”), [0031] (“floating-point multiplication is performed in a series of stages ... one partial product is generated for every bit in the multiplier 404, a partial product 420-i being

generated during a corresponding stage 410-i. If the value of the multiplier is 0, its corresponding partial product consists only of 0s; if the value of the bit is 1, its corresponding partial product is a copy of the multiplicand.”).

486. **Step 3:** For each bit in the multiplier,

- a. Compute a partial product that equals 0 (when the multiplier bit is 0) or the multiplicand (when the multiplier bit is 1). *See* Dockser, [0031] (“one partial product is generated for every bit in the multiplier 404, a partial product 420-i being generated during a corresponding stage 410-i. If the value of the multiplier is 0, its corresponding partial product consists only of 0s; if the value of the bit is 1, its corresponding partial product is a copy of the multiplicand.”).
- b. Left-shift the partial product by inserting a number of 0s, at the partial products’s right end, equal to the multiplier bit’s bit position (with the right-most multiplier bit treated as having bit position 0, such that the first partial product is not left-shifted at all). *See* Dockser, [0031] (“Each partial product 420-i is left-shifted, as a function of the multiplier bit with which it is associated, after which the operation moves on to the next stage. Each partial product can thus be viewed as a shifted number. The partial product associated with bit 0 in the

multiplier is left-shifted zero bits, and the partial product associated with bit 1 is left-shifted one bit.”).

- c. Convert to 0 all partial product bits less significant than the “selected subprecision.” Dockser, [0032] (“in FIG. 3B, the selection of a desired reduced precision by the controller 130 is indicated with a line 405. As in the case of floating-point addition ... *power may be removed from the logic used to implement the stages to the right of the line 405.* ... In FIG. 3B, the bits provided to the powered on logic are shown as Xs, while *the bits provided to the powered down stages are shown as circles*”), [0033] (“for the first partial product 420-1, the logic for a number of bits N, shown using reference numeral 402, is unpowered. For the second partial product, the logic for N-1 bits is unpowered, and so forth”), Fig. 3B (showing partial product bits to the right of “SELECTED PRECISION” line 405 unpowered). As I explain in Section VI.B.4.c(2) above, a POSA would have understood Dockser to disclose that the bits from which power is removed in the logic are dropped to 0s, or alternatively this would have been the conventional and obvious way to implement the power removal Dockser describes.


- d. Add the result of step c to the output value. Dockser, [0031] (“The partial products or shifted floating-point numbers 420-i are added together to generate the output value 430 for the multiplication.”), [0034] (referring to “The output value, resulting from the floating-point multiplication described above” and “The output value 430”).
- e. (Repeat step 3 for each bit in the multiplier.) *See* Dockser, [0031 (“one partial product is generated for every bit in the multiplier 404 ... The partial products or shifted floating-point numbers 420-i are added together to generate the output value 430 for the multiplication.”).

487. **Step 4:** Interpret the generated sum as a binary output number Q' in which the radix point is to the left of the rightmost 46 bits of the generated sum. I explain how my program accomplishes this in paragraphs 488-490 below.

488. As I explained in paragraph 485 above, for the purposes of computing and adding the partial products to produce an output value, my program interprets the mantissa bits (including the hidden bit) of the operands as integers. This means that the sum of the partial products that is generated in step 3 is also represented as an integer during the software program’s computations of step 3. But it mathematically represents an output number Q' , which could be up to 48 bits long, which has a fractional part consisting of the right-most 46 bits.

489. To see why this is so, consider the example of how a human would perform pencil-and-paper long multiplication of two decimal numbers with fractional parts, *e.g.* 9.99×8.88 . As shown below, one way of thinking of such long multiplication is to treat the numbers as integers, compute the partial products (left-shifting as needed), sum them, and then place the decimal point in the final product such that the number of digits to the right of the decimal point is equal to the total number of digits to the right of the decimal point in the two numbers being multiplied. In this example, there are 4 total digits to the right of the decimal point in the two numbers being multiplied (9.99 and 8.88).

				9.	9	9	Operand A
				8.	8	8	Operand B
				<hr/>			
			7	9	9	2	} partial products
		7	9	9	2	0	
(+)	7	9	9	2	0	0	
		<hr/>					
	8	8.	7	1	1	2	Products


Decimal point

490. A POSA would have understood that, when the output value of multiplying two 24-bit mantissas is 48 bits wide, then treating it as a mantissa with 46 bits to the *right* of the radix point would cause it to have two bits to the *left* of the radix point, which would put it outside of the “normal” mantissa range of $1 \leq$

mantissa < 2. See, e.g., Dockser-Lall, [0007] (explaining in the case of a “hypothetical 5-bit [mantissa]” that “The product of two normal significands may assume a value in the range [1,4)—that is, from exactly one to almost four. This intermediate product significand thus may assume the form 1.fraction or 1x.fraction, the latter for values from two to almost four (10.0000 to 11.1111).”). To account for this, a POSA would have understood that “[f]loating-point multipliers adjust this intermediate result by shifting the binary point left and incrementing the exponent by one, *as a routine incident of floating-point multiplication.*” Dockser-Lall, [0007]. A POSA would have understood that this process was sometimes known as “normalization,” and Dockser discloses including a “normalizer” in its multiplier. Dockser, [0022] (“the multiplier 144 may ... include one or more *well-known conventional subunits* such as aligners that align input operands, *normalizers that shift the result into standard format* ...”). To perform the normalization operation, my program analyzes the output value resulting after Step 3 above (see paragraph 486 above), which could be as long as 48 bits wide (see paragraph 453 above), finds the *left-most* (i.e. most significant) bit that is a ‘1,’ and then and *right-shifts* the output value a sufficient number of bits so that this left-most ‘1’ bit is in the 24th bit position (counting from the right). The number of bits by which the output value is right-shifted depends on the bit position of the left-most ‘1’ bit. If the left-most ‘1’ is in the 47th position

(counting from the right), then the number is right-shifted 23 bits, which causes the left-most ‘1’ bit to now appear in the 24th bit position (because $47 - 23 = 24$). On the other hand, if the left-most ‘1’ bit is in the 48th bit position, then the output value is right-shifted 24 bits, which causes the left-most ‘1’ bit to now appear in the 24th bit position (because $48 - 24 = 24$). A POSA would have understood that this process normalizes the output value, because following the right-shift, the 24 right-most bits of the output value can be viewed as a 24-bit mantissa in which the 23 right-most bits represent the fraction portion and the 24th bit (which is guaranteed to be a 1) represents the “hidden” integer bit (*e.g.*, a 24-bit sequence 1001110000011111111110000111100111100111011001010 is treated as a mantissa 1.001110000011111111110000111100111100111011001010).

491. My program then uses the normalized single-precision mantissa generated using the process I discussed in the previous paragraph as the mantissa of a single-precision floating-point number Q' with ‘0’ sign bit and an exponent value of either 0 or 1, depending on how many bits the mantissa product was right-shifted. To do this, the program copies the 23 right-most bits of the right-shifted mantissa into the fraction bit field of a floating-point number; the 24th bit does not need to be copied because it represents the “hidden” bit that is a 1 in normal floating-point numbers. The sign bit is set to 0 because both of operands A and B are positive. *See* paragraph 477 above. The exponent value of Q' is either 0 or 1,

depending on number of bits that the mantissa multiplier output value was right-shifted in the “normalization” process I discussed in the previous paragraph. If the mantissa multiplier output value was right-shifted 23 bits, then the exponent is 0, but if it was right-shifted 24 bits, then the exponent is 1. A POSA would have understood that in the latter case, incrementing the exponent value to 1 does not change the value of the floating-point number because it is mathematically cancelled out by the right-shifting of the mantissa one extra bit (*i.e.* 24 bits instead of 23). Therefore, mathematically, the value of the single-precision floating-point number Q' is the value of interpreting the sum of partial products generated by the mantissa multiplier as a binary number in which the radix point is to the left of the rightmost 46 bits of the generated sum.²

² When a selected precision level retaining, *e.g.*, 9 fraction bits in the output sum of partial products is applied, the single-precision floating-point version of the number Q' may have either 9 or 10 non-zeroed fraction bits after the normalization I discuss in paragraph 490 above. A POSA would have understood that Dockser’s techniques allow for either retaining the 10th-most-significant fraction bit if non-zero, or truncating it to zero. My software program tested Dockser’s techniques both ways, and confirmed that both meet the ’273 patent’s claimed X and Y ranges. The results I report in the tables in this Section are those that retain the 10th

492. I wrote my logic bit-dropping software program using C and CUDA and parallelized it on an NVidia Quadro GV100 GPU, just as I did with my register bit-dropping program, as I discussed in Appendix I.B above. This would have been within the level of ordinary skill of the POSA. In my program, both the selected precision level applied in the multiplier and the Y percentage being tested are defined in the source code. To test different selected precision levels and different minimum Y percentages, I changed those values and then re-compiled and re-ran my program. An exemplary version of the source code for my program testing $Y=0.05\%$ with a selected precision of 9 fraction bits is attached to my declaration as Attachment B1.

493. The results from my program shown in the table below demonstrate that Dockser's logic bit-dropping technique produces the claimed minimum relative error percentage Y for more than the claimed minimum percentage X of

bit (rather than truncating it), because that produces the smaller error and more conservatively meets the '273 patent's claims. Demonstrating that Dockser's techniques when retaining that 10th bit meet the claims also demonstrate that Dockser's techniques when truncating that 10th bit meet the claims, because the relative error when truncating that 10th bit is at least as large as when truncating that 10th bit.

the possible valid inputs at precision levels retaining 9 (Dockser, [0026]) or 5 (Tong, 282) fraction bits in the output, for each claimed X/Y combination. As I did in Appendix I.B above when reporting the results for my register bit-dropping program, in the table below I report both not only the percentage of tested pairs that meet the claimed Y percentage as output by my software program, but also a “Dockser’s X” percentage that is conservatively adjusted to exclude possible overflow/underflow pairs, by multiplying the percentage output by my software program by the adjustment ratio discussed in Appendix I.E below. The “Dockser’s X” percentages, as the table below shows, are greater than the minimum X percentage recited in any of the ’273 patent’s claims, thus meeting the claims.

Retained Fraction Bits	Claimed Y%	Percentage of Tested Pairs Meeting Claimed Y%	Adjusted “Dockser’s X%”
9 fraction bits	≥ 0.05%	99.873716%	≥ 99.35080%
	≥ 0.10%	98.837578%	≥ 98.32009%
	≥ 0.15%	94.862501%	≥ 94.36582%
	≥ 0.20%	85.437855%	≥ 84.99052%
5 fraction bits	≥ 0.05%	99.997444%	≥ 99.47388%
	≥ 0.10%	99.989768%	≥ 99.46625%
	≥ 0.15%	99.976929%	≥ 99.45347%
	≥ 0.20%	99.958818%	≥ 99.43546%

494. My program stores the exact mathematical calculation of the product $Q = A \times B$ in IEEE-754 double-precision floating-point format for the same reasons I discussed in Appendix I.B (paragraphs 453-454) above. If Singular were

to argue (incorrectly) that the “exact mathematical calculation” of multiplying two single-format floating-point operands is a single-format product (having a 23-bit-fraction mantissa), the claims would still be met. To confirm this, I wrote an alternative version of my software program for logic bit-dropping (attached to this declaration as Attachment B2) that performed the same steps I discussed in paragraphs 476-491 above, except that it stores the product $Q = A \times B$ as a single-format (single-precision, having only up to 23 mantissa fraction bits) number. The table below reports the same percentages as the table I explained in paragraph 493 above, but using the alternate version of my program discussed in this paragraph. As the table below shows, my test using this alternative produced identical results up to four decimal places of the “Dockser X” percentage as my test in paragraph 493 above that used the double-precision product as the exact mathematical calculation.

Retained Fraction Bits	Claimed Y%	Percentage of Tested Pairs Meeting Claimed Y%	Adjusted “Dockser’s X%”
9 fraction bits	$\geq 0.05\%$	99.873716%	$\geq 99.35080\%$
	$\geq 0.10\%$	98.837572%	$\geq 98.32008\%$
	$\geq 0.15\%$	94.862499%	$\geq 94.36582\%$
	$\geq 0.20\%$	85.437856%	$\geq 84.99052\%$
5 fraction bits	$\geq 0.05\%$	99.997444%	$\geq 99.47388\%$
	$\geq 0.10\%$	99.989768%	$\geq 99.46625\%$
	$\geq 0.15\%$	99.976929%	$\geq 99.45347\%$
	$\geq 0.20\%$	99.958817%	$\geq 99.43546\%$

E. Adjustment to Account for Overflow/Underflow

495. In normal IEEE-754 single-format numbers, the possible values of the exponent range from -126 to 127. *See* Dockser-Lall, [0004] (“In the IEEE 754 standard, the value of the exponent for a single-precision floating-point number ranges from -126 to 127.”). As I explained in paragraph 234 above, in floating-point multiplication, the exponents of the two numbers being multiplied are added. This means that mathematically, the product of two floating-point numbers whose exponents are both in the -126 to 127 range could possibly have an exponent whose value falls outside of that range. For example, $-100 + -100 = -200$, or $100 + 100 = 200$. A POSA would have understood that when a multiplier produces an output that is too large to be representable in the numerical representation the multiplier uses for its output, this condition is referred to as an “overflow,” while an output that is too small to be representable is a condition referred to as an “underflow.” In the IEEE-754 standard, these conditions are treated as exception conditions, and the multiplier does not produce an output representing a numerical value. Because the multiplier does not produce an output representing a numerical value in these cases, the pair of operands in these cases is not a “valid input” because executing the first operation on that input does not “produce a first output signal representing a second numerical value” as the independent claims of the ’273 patent recite. Thus, for the purposes of my

analysis, I exclude from the set of possible valid inputs any pairs of normal single-precision floating point numbers that could produce an overflow or underflow.

496. For certain pairs of normal IEEE-754 single-format floating-point numbers, whether an overflow or underflow occurs depends only on the exponent values of the operands and is independent of their mantissa values. In particular, if two normal single-format floating-point numbers A and B have exponents E_A and E_B , respectively, where their sum $E_A + E_B \geq 128$, their product will overflow, because the sum of their exponents is greater than 127 and the product of their mantissas is at least 1 (since they are both normal numbers). Likewise, if two normal single-precision floating-point numbers A and B have exponent values E_A and E_B , respectively whose sum $E_A + E_B \leq -128$, their product will underflow, because even if the exponent of the product is incremented as a result of the floating-point multiplier normalizing the output (*see my explanation in Appendix I.D above*), it would still be -127, which is less than -126.

497. There are 254 possible values for the exponent of a normal IEEE-754 single-format floating-point number (-126 to 127, including 0). Therefore, there are $254 \times 254 = 64,516$ possible pairs of exponent values represented among the possible pairs of normal IEEE-754 single-format floating-point numbers. Of those 64,516 pairs of exponent values, there are 16,003 pairs whose sum is either ≥ 128 or ≤ -128 , such that their product will overflow or underflow; thus, I exclude all

pairs whose exponents are in this category from the set of possible valid inputs. I determined that there are 16,003 such pairs by writing and running a simple program in the C programming language, attached to this Declaration as Attachment C1. Excluding these pairs from the set of possible valid inputs has no mathematical effect on my analysis of whether Dockser meets the '273 patent's claims. As I discussed in Appendix I.B-I.D. above, my analyses determine the percentage of possible mantissa pairs that produce at least the relative error amount recited in the claims. This determines the percentage of all possible floating-point number pairs that produces at least the relative error amount recited in the claims, because the set of all possible pairs of floating-point-numbers pairs every possible mantissa pair with every possible exponent pair, meaning that the number of possible exponent pairs appears in both the numerator and denominator of the calculation of the percentage of floating-point number pairs that produce at least the recited error amount, and is thus cancelled out in the percentage calculation, as shown below. If the exponent pairs that will cause an overflow or underflow are excluded from the set of possible valid inputs, that reduces the number of "possible exponent pairs" by the same amount in both the numerator and the denominator of the percentage calculation below, such that the effect cancels out and the percentage is unchanged.

(percentage of FP number pairs producing required error)

$$= \frac{(\text{mantissa pairs producing required error}) \times (\text{possible exponent pairs})}{(\text{total mantissa pairs}) \times (\text{possible exponent pairs})} \times 100$$

498. Because there are 16,003 exponent pairs that will lead to overflow or underflow (because their sum is either ≥ 128 or ≤ -128), as I explained in the previous paragraph, excluding those exponent pairs from the exponent pairs in the set of possible valid inputs leaves $64,516 - 16,003 = 48,513$ possible pairs of exponent values in the set of possible valid inputs.

499. Of those remaining pairs, there are two sets of pairs for which overflow or underflow *could* occur depending on the values of the mantissas of the operands. First, if the sum of the exponents of the two floating-point numbers A and B is 127, then an overflow will occur *if* the mantissa product is greater than or equal to 2, because normalization will increment the exponent to 128. Second, if the sum of the exponents is -127, then overflow will occur if the mantissa product is less than 2, because if the mantissa product is greater than 2, then normalization will increment the exponent sum to -126. There are 254 pairs of exponents whose sum is either -126 or 127. I determined that there are 254 such pairs by writing and running the simple program in the C programming language attached as Attachment C1.

500. Because the pairs of normal IEEE-754 single-format floating-point numbers whose exponents sum to either -126 or 127 *may or may not* overflow or underflow (depending on the values of their mantissas), I did not exclude them from the set of possible valid inputs. Instead, in order to be conservative, I assumed for the purposes of my analysis that all pairs of normal IEEE-754 single-format floating-point numbers whose exponents add up to either -126 or 127 *are* in the set of “possible valid inputs” as claimed, but that *none* of those pairs of floating-point numbers produce a product whose value differs from the “exact mathematical calculation” by the Y% required in any of the ’273 patent’s claims. The reason this is a conservative approach is that it *reduces* the percentage of the possible valid inputs that are reported as producing the claimed relative error amount below that percentage’s true value, because it increases the number of possible valid inputs in the percentage calculation (making the denominator larger) while potentially excluding possible valid inputs that would have produced the claimed relative error amount (making the numerator smaller). To adjust my reported “Dockser’s X” percentages in accordance with this assumption, I multiply the percentage of mantissa pairs that my software program outputs as producing the claimed relative error amount Y by the quantity $(48259 \div 48513)$, which is the ratio of the number of pairs of exponent values that *cannot* produce an overflow or underflow over the number of pairs of exponent values that I

calculated in paragraph 498 above as being exponent pairs that are not guaranteed to always overflow or underflow. The value of this multiplier (rounded to 8 decimal places) is .9947629. The fact that this multiplier is very close to 1 reflects that very few of all possible input pairs could possibly produce overflow or underflow; thus conservatively excluding those input pairs has very little effect on the X percentage of possible valid inputs that meets the claimed relative error amount Y.

I declare that all statements made herein of my own knowledge are true, that all statements made on information and belief are believed to be true, and that these statements were made with the knowledge that willful false statements and the like are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code.

I declare under penalty of perjury that the foregoing is true and correct.

Dated: November 6, 2020

A handwritten signature in dark ink, appearing to read "Charles W. Smith", is written over a horizontal line.

ATTACHMENT A1

```

1  #include <stdint.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include <time.h>
5  #include <cuda_runtime.h>
6  #include <helper_cuda.h>
7
8
9  // error threshold
10 // #define threshold (0.0005)
11 // #define operandBits (9)
12 // #define resultBits (9)
13
14 // error threshold
15 #define threshold (0.0005)
16 #define operandBits (9)
17 #define resultBits (9)
18
19 // CUDA parameters
20 #define NUM_THREADS 1024
21 #define NUM_BLOCKS 8192
22
23 // truncate bits
24 #define operandTruncMask (0xffffffff<<(23-operandBits))
25 #define resultTruncMask (0xffffffff<<(23-resultBits))
26
27 // ieee754 1.0
28 #define signExp (0x3f800000)
29
30 // access value as int32 or float
31 union rawFloat {
32     uint32_t fixed;
33     float ieee;
34 };
35
36 __global__ void
37 fpMult(uint32_t *opr_result, uint32_t *res_result) {
38     // operand mantissas
39     uint32_t mantissa1;
40     uint32_t mantissa2;
41
42     // full ieee754 operands
43     union rawFloat operand1;
44     union rawFloat operand2;
45
46     // truncated ieee754
47     union rawFloat truncate1;
48     union rawFloat truncate2;
49
50     // full double ieee754 result
51     double result;
52
53     // result with truncated operands
54     union rawFloat operandResult;
55
56     // truncated result
57     union rawFloat singleResult;
58     union rawFloat truncResult;
59
60     // truncated operand difference
61     double deltaOperand;
62
63     // truncated result difference
64     double deltaResult;
65
66     // count of values greater than threshold

```



```

67  uint32_t operandCount;
68  uint32_t resultCount;
69
70  // index into count array
71  uint32_t index;
72
73  // cycle through all mantissas for operand 1
74  index = (blockDim.x * blockIdx.x) + threadIdx.x;
75  mantissa1 = index;
76
77  // build the untruncated and truncated first operand
78  operand1.fixed = signExp | mantissa1;
79  truncate1.fixed = operand1.fixed & operandTruncMask;
80
81  operandCount = 0;
82  resultCount = 0;
83
84  // all ieee floating point values for second operand >= 1.0 and < 2.0
85  for (mantissa2 = 0; mantissa2 < (1 << 23); mantissa2++) {
86
87      // build the untruncated and truncated operand 2
88      operand2.fixed = signExp | mantissa2;
89      truncate2.fixed = operand2.fixed & operandTruncMask;
90
91      // reference ieee754 double precision full result
92      result = (double)operand1.ieee * (double)operand2.ieee;
93
94      // result with truncated operands
95      operandResult.ieee = truncate1.ieee * truncate2.ieee;
96
97      // truncated result
98      singleResult.ieee = operand1.ieee * operand2.ieee;
99      truncResult.fixed = singleResult.fixed & resultTruncMask;
100
101      // calculate operand truncation difference as percentage of full result
102      deltaOperand = result - (double)operandResult.ieee;
103      deltaOperand = fabs(deltaOperand);
104      deltaOperand = deltaOperand / result;
105
106      // calculate result truncation difference as percentage of full result
107      deltaResult = result - (double)truncResult.ieee;
108      deltaResult = fabs(deltaResult);
109      deltaResult = deltaResult / result;
110
111      // if over threshold bump operand truncation count
112      if (deltaOperand >= threshold) { operandCount++; }
113
114      // if over threshold bump result truncation count
115      if (deltaResult >= threshold) { resultCount++; }
116
117  }
118  // copy values out to GPU memory
119  opr_result[index] = operandCount;
120  res_result[index] = resultCount;
121 }
122
123 int main(int argc, char** argv) {
124
125     printf("threshold = %f operand bits = %d result bits = %d\n",
126           threshold, operandBits, resultBits);
127     fflush(stdout);
128
129     printf("Allocating host memory\n");
130     fflush(stdout);
131
132     time_t startTime = time(NULL);

```

```

133 // allocate partial count arrays for output on host
134 uint32_t* host_opr_result = (uint32_t *)malloc(NUM_BLOCKS * NUM_THREADS *
135 sizeof(uint32_t));
136 if (host_opr_result == NULL) {
137     fprintf(stderr, "Failed to allocate host array!\n");
138     exit(EXIT_FAILURE);
139 }
140 uint32_t* host_res_result = (uint32_t*)malloc(NUM_BLOCKS * NUM_THREADS *
141 sizeof(uint32_t));
142 if (host_res_result == NULL) {
143     fprintf(stderr, "Failed to allocate host array!\n");
144     exit(EXIT_FAILURE);
145 }
146 printf("Allocating gpu memory\n");
147 fflush(stdout);
148
149 // Error code to check return values for CUDA calls
150 cudaError_t err = cudaSuccess;
151
152 // allocate partial count arrays for output on GPU
153 uint32_t* gpu_opr_result;
154 err = cudaMalloc((void **)&gpu_opr_result, NUM_BLOCKS * NUM_THREADS *
155 sizeof(uint32_t));
156 if (err != cudaSuccess) {
157     fprintf(stderr, "Failed to allocate gpu array!\n");
158     exit(EXIT_FAILURE);
159 }
160 uint32_t* gpu_res_result;
161 err = cudaMalloc((void **)&gpu_res_result, NUM_BLOCKS * NUM_THREADS *
162 sizeof(uint32_t));
163 if (err != cudaSuccess) {
164     fprintf(stderr, "Failed to allocate gpu array!\n");
165     exit(EXIT_FAILURE);
166 }
167 printf("Fire off GPU kernel\n");
168 fflush(stdout);
169
170 // fire off the CUDA kernel
171 fpMult << <NUM_BLOCKS, NUM_THREADS >> > (gpu_opr_result, gpu_res_result);
172 err = cudaGetLastError();
173 if (err != cudaSuccess)
174 {
175     fprintf(stderr, "Failed to launch vectorAdd kernel (error code %s)!\n",
176         cudaGetErrorString(err));
177     exit(EXIT_FAILURE);
178 }
179 printf("Copy results from GPU to host\n");
180 fflush(stdout);
181
182 // Copy the device result vectors in device memory to the host result vector
183 // in host memory.
184 err = cudaMemcpy(host_opr_result, gpu_opr_result, NUM_BLOCKS * NUM_THREADS *
185 sizeof(uint32_t), cudaMemcpyDeviceToHost);
186 if (err != cudaSuccess)
187 {
188     fprintf(stderr, "Failed to copy vector from device to host (error code %s)!\n",
189         cudaGetErrorString(err));
190     exit(EXIT_FAILURE);
191 }
192
193 err = cudaMemcpy(host_res_result, gpu_res_result, NUM_BLOCKS * NUM_THREADS *
194 sizeof(uint32_t), cudaMemcpyDeviceToHost);

```

```

191 if (err != cudaSuccess)
192 {
193     fprintf(stderr, "Failed to copy vector from device to host (error code %s)!\n",
194         cudaGetErrorString(err));
195     exit(EXIT_FAILURE);
196 }
197 uint64_t truncOperandCount = 0LL;
198 uint64_t truncResultCount = 0LL;
199
200 printf("Sum up results\n");
201 fflush(stdout);
202
203 // sum up to create total count
204 for (int i = 0; i < (NUM_BLOCKS * NUM_THREADS); i++) {
205     truncOperandCount += host_opr_result[i];
206     truncResultCount += host_res_result[i];
207 }
208
209 double numPasses = ((double)NUM_BLOCKS) * ((double)NUM_THREADS) * (double)(1LL <<
210 23);
211
212 printf("%lld seconds\n", time(NULL)-startTime);
213
214 printf("operand truncation %lld/%lld = %f%%\n",
215     truncOperandCount,
216     (uint64_t)numPasses,
217     ((double)truncOperandCount / numPasses)*100.0);
218 printf("result truncation %lld/%lld = %f%%\n",
219     truncResultCount,
220     (uint64_t)numPasses,
221     ((double)truncResultCount / numPasses)*100.0);
222
223 // free GPU memory
224 err = cudaFree(gpu_opr_result);
225 if (err != cudaSuccess)
226 {
227     fprintf(stderr, "Failed to free device operand vector (error code %s)!\n",
228         cudaGetErrorString(err));
229     exit(EXIT_FAILURE);
230 }
231 err = cudaFree(gpu_res_result);
232 if (err != cudaSuccess)
233 {
234     fprintf(stderr, "Failed to free device result vector (error code %s)!\n",
235         cudaGetErrorString(err));
236     exit(EXIT_FAILURE);
237 }
238
239 // Free host memory
240 free(host_opr_result);
241 free(host_res_result);
242 }

```

ATTACHMENT A2

```

1  #include <stdint.h>
2  #include <stdint.h>
3  #include <stdlib.h>
4  #include <math.h>
5  #include <time.h>
6  #include <cuda_runtime.h>
7  #include <helper_cuda.h>
8
9  // error threshold
10 // #define threshold (0.0005)
11 // #define operandBits (9)
12 // #define resultBits (9)
13
14 // error threshold
15 #define threshold (0.0005)
16 #define operandBits (9)
17 #define resultBits (9)
18
19 // CUDA parameters
20 #define NUM_THREADS 1024
21 #define NUM_BLOCKS 8192
22
23 // truncate bits
24 #define operandTruncMask (0xffffffff<<(23-operandBits))
25 #define resultTruncMask (0xffffffff<<(23-resultBits))
26
27 // ieee754 1.0
28 #define signExp (0x3f800000)
29
30 // access value as int32 or float
31 union rawFloat {
32     uint32_t fixed;
33     float ieee;
34 };
35
36 __global__ void
37 fpMult(uint32_t *opr_result, uint32_t *res_result) {
38     // operand mantissas
39     uint32_t mantissa1;
40     uint32_t mantissa2;
41
42     // full ieee754 operands
43     union rawFloat operand1;
44     union rawFloat operand2;
45
46     // truncated ieee754
47     union rawFloat truncate1;
48     union rawFloat truncate2;
49
50     // single ieee754 result
51     float result;
52
53     // result with truncated operands
54     union rawFloat operandResult;
55
56     // truncated result
57     union rawFloat singleResult;
58     union rawFloat truncResult;
59
60     // truncated operand difference
61     double deltaOperand;
62
63     // truncated result difference
64     double deltaResult;
65
66     // count of values greater than threshold

```

```

67  uint32_t operandCount;
68  uint32_t resultCount;
69
70  // index into count array
71  uint32_t index;
72
73  // cycle through all mantissas for operand 1
74  index = (blockDim.x * blockIdx.x) + threadIdx.x;
75  mantissa1 = index;
76
77  // build the untruncated and truncated first operand
78  operand1.fixed = signExp | mantissa1;
79  truncate1.fixed = operand1.fixed & operandTruncMask;
80
81  operandCount = 0;
82  resultCount = 0;
83
84  // all ieee floating point values for second operand >= 1.0 and < 2.0
85  for (mantissa2 = 0; mantissa2 < (1 << 23); mantissa2++) {
86
87      // build the untruncated and truncated operand 2
88      operand2.fixed = signExp | mantissa2;
89      truncate2.fixed = operand2.fixed & operandTruncMask;
90
91      // reference ieee754 single precision result
92      result = operand1.ieee * operand2.ieee;
93
94      // result with truncated operands
95      operandResult.ieee = truncate1.ieee * truncate2.ieee;
96
97      // truncated result
98      singleResult.ieee = operand1.ieee * operand2.ieee;
99      truncResult.fixed = singleResult.fixed & resultTruncMask;
100
101      // calculate operand truncation difference as percentage of full result
102      deltaOperand = (double)result - (double)operandResult.ieee;
103      deltaOperand = fabs(deltaOperand);
104      deltaOperand = deltaOperand / result;
105
106      // calculate result truncation difference as percentage of full result
107      deltaResult = (double)result - (double)truncResult.ieee;
108      deltaResult = fabs(deltaResult);
109      deltaResult = deltaResult / result;
110
111      // if over threshold bump operand truncation count
112      if (deltaOperand >= threshold) { operandCount++; }
113
114      // if over threshold bump result truncation count
115      if (deltaResult >= threshold) { resultCount++; }
116
117  }
118  // copy values out to GPU memory
119  opr_result[index] = operandCount;
120  res_result[index] = resultCount;
121 }
122
123 int main(int argc, char** argv) {
124
125     printf("threshold = %f operand bits = %d result bits = %d\n",
126           threshold, operandBits, resultBits);
127     fflush(stdout);
128
129     printf("Allocating host memory\n");
130     fflush(stdout);
131
132     time_t startTime = time(NULL);

```

```

133 // allocate partial count arrays for output on host
134 uint32_t* host_opr_result = (uint32_t *)malloc(NUM_BLOCKS * NUM_THREADS *
135 sizeof(uint32_t));
136 if (host_opr_result == NULL) {
137     fprintf(stderr, "Failed to allocate host array!\n");
138     exit(EXIT_FAILURE);
139 }
140 uint32_t* host_res_result = (uint32_t*)malloc(NUM_BLOCKS * NUM_THREADS *
141 sizeof(uint32_t));
142 if (host_res_result == NULL) {
143     fprintf(stderr, "Failed to allocate host array!\n");
144     exit(EXIT_FAILURE);
145 }
146 printf("Allocating gpu memory\n");
147 fflush(stdout);
148
149 // Error code to check return values for CUDA calls
150 cudaError_t err = cudaSuccess;
151
152 // allocate partial count arrays for output on GPU
153 uint32_t* gpu_opr_result;
154 err = cudaMalloc((void **)&gpu_opr_result, NUM_BLOCKS * NUM_THREADS *
155 sizeof(uint32_t));
156 if (err != cudaSuccess) {
157     fprintf(stderr, "Failed to allocate gpu array!\n");
158     exit(EXIT_FAILURE);
159 }
160 uint32_t* gpu_res_result;
161 err = cudaMalloc((void **)&gpu_res_result, NUM_BLOCKS * NUM_THREADS *
162 sizeof(uint32_t));
163 if (err != cudaSuccess) {
164     fprintf(stderr, "Failed to allocate gpu array!\n");
165     exit(EXIT_FAILURE);
166 }
167 printf("Fire off GPU kernel\n");
168 fflush(stdout);
169
170 // fire off the CUDA kernel
171 fpMult << <NUM_BLOCKS, NUM_THREADS >> > (gpu_opr_result, gpu_res_result);
172 err = cudaGetLastError();
173 if (err != cudaSuccess)
174 {
175     fprintf(stderr, "Failed to launch vectorAdd kernel (error code %s)!\n",
176         cudaGetErrorString(err));
177     exit(EXIT_FAILURE);
178 }
179 printf("Copy results from GPU to host\n");
180 fflush(stdout);
181
182 // Copy the device result vectors in device memory to the host result vector
183 // in host memory.
184 err = cudaMemcpy(host_opr_result, gpu_opr_result, NUM_BLOCKS * NUM_THREADS *
185 sizeof(uint32_t), cudaMemcpyDeviceToHost);
186 if (err != cudaSuccess)
187 {
188     fprintf(stderr, "Failed to copy vector from device to host (error code %s)!\n",
189         cudaGetErrorString(err));
190     exit(EXIT_FAILURE);
191 }
192
193 err = cudaMemcpy(host_res_result, gpu_res_result, NUM_BLOCKS * NUM_THREADS *
194 sizeof(uint32_t), cudaMemcpyDeviceToHost);

```

```

191 if (err != cudaSuccess)
192 {
193     fprintf(stderr, "Failed to copy vector from device to host (error code %s)!\n",
194         cudaGetErrorString(err));
195     exit(EXIT_FAILURE);
196 }
197 uint64_t truncOperandCount = 0LL;
198 uint64_t truncResultCount = 0LL;
199
200 printf("Sum up results\n");
201 fflush(stdout);
202
203 // sum up to create total count
204 for (int i = 0; i < (NUM_BLOCKS * NUM_THREADS); i++) {
205     truncOperandCount += host_opr_result[i];
206     truncResultCount += host_res_result[i];
207 }
208
209 double numPasses = ((double)NUM_BLOCKS) * ((double)NUM_THREADS) * (double)(1LL <<
210 23);
211
212 printf("%lld seconds\n", time(NULL)-startTime);
213
214 printf("operand truncation %lld/%lld = %f%%\n",
215     truncOperandCount,
216     (uint64_t)numPasses,
217     ((double)truncOperandCount / numPasses)*100.0);
218 printf("result truncation %lld/%lld = %f%%\n",
219     truncResultCount,
220     (uint64_t)numPasses,
221     ((double)truncResultCount / numPasses)*100.0);
222
223 // free GPU memory
224 err = cudaFree(gpu_opr_result);
225 if (err != cudaSuccess)
226 {
227     fprintf(stderr, "Failed to free device operand vector (error code %s)!\n",
228         cudaGetErrorString(err));
229     exit(EXIT_FAILURE);
230 }
231 err = cudaFree(gpu_res_result);
232 if (err != cudaSuccess)
233 {
234     fprintf(stderr, "Failed to free device result vector (error code %s)!\n",
235         cudaGetErrorString(err));
236     exit(EXIT_FAILURE);
237 }
238
239 // Free host memory
240 free(host_opr_result);
241 free(host_res_result);
242 }

```


ATTACHMENT B1

```

1  #include <stdint.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include <time.h>
5  #include <cuda_runtime.h>
6  #include <helper_cuda.h>
7
8
9  // number of active bits
10 // #define inactiveBits (37)
11 // #define activeBits (46 - inactiveBits)
12 // error threshold
13 // #define threshold (0.0005)
14
15 // number of inactive bits
16 #define inactiveBits (37)
17 #define activeBits (46 - inactiveBits)
18 // error threshold
19 #define threshold (0.0005)
20
21 // CUDA parameters
22 #define NUM_THREADS 1024
23 #define NUM_BLOCKS 8192
24
25 // mask out the inactive LSB from the accumulator
26 #define inactiveMask (0xffffffffffffffffLL<<inactiveBits)
27 // mask out the inactive LSB from the result after normalization
28 #define normMask (0xffffffff<<(inactiveBits-23))
29
30 // ieee754 1.0
31 #define signExp (0x3f800000)
32 // lsb of exponent in ieee754
33 #define firstExpBit (0x800000)
34
35 // access value as int32 or float
36 union rawFloat {
37     uint32_t fixed;
38     float ieee;
39 };
40
41 __global__ void
42 fpMult(uint32_t *mult_result, uint32_t *norm_result) {
43
44     // operand mantissas
45     uint32_t mantissa1;
46     uint32_t mantissa2;
47
48     // full ieee754 operands
49     union rawFloat operand1;
50     union rawFloat operand2;
51
52     // full double ieee754 result
53     double result;
54     // ieee754 result with inactive bits
55     union rawFloat inactiveResult;
56
57     // truncated result difference
58     double deltaResult;
59
60     // index into operand2 bits
61     int bit;
62
63     // 64 bit shift and add accumulator
64     uint64_t accum;
65
66     // operand mantissas with implicit 1. added

```

```

67  uint32_t mult1;
68  uint32_t mult2;
69
70  // count of error over threshold for multiplier truncation
71  uint32_t multCount;
72
73  // count of error over threshold for truncation after normalization
74  uint32_t normCount;
75
76  // index into count array
77  uint32_t index;
78
79  // cycle through all mantissas for operand 1
80  index = (blockDim.x * blockIdx.x) + threadIdx.x;
81  mantissa1 = index;
82
83  // build the first operand
84  operand1.fixed = signExp | mantissa1;
85
86  multCount = 0;
87  normCount = 0;
88
89  // all ieee floating point values for second operand >= 1.0 and < 2.0
90  for(mantissa2 = 0; mantissa2 < (1<<23); mantissa2++) {
91
92      // build the second operand
93      operand2.fixed = signExp | mantissa2;
94
95      // reference ieee754 double precision full result
96      result = (double)operand1.ieee * (double)operand2.ieee;
97
98      // extract the mantissa
99      mult1 = mantissa1;
100     mult2 = mantissa2;
101
102     // set the implicit 1.
103     mult1 |= firstExpBit;
104     mult2 |= firstExpBit;
105
106     //
107     // multiply implemented as shift and add
108     //
109     accum = 0LL;
110
111     // iterate over 24 bit mantissa value
112     for(bit=0;bit<24;bit++) {
113
114         // for each bit position in mantissa1 add shifted mantissa2
115         if((mult1>>bit)&1) {
116             accum += (((uint64_t)mult2) << bit) & inactiveMask;
117         }
118     }
119
120     // normalize result
121     if(accum >> 47) {
122
123         // shift accumulated value by 24 bits
124         accum = accum >> 24;
125         inactiveResult.fixed = accum;
126         inactiveResult.fixed |= signExp;
127
128         // increment the exponent
129         inactiveResult.fixed += firstExpBit;
130
131     } else {

```

```

133 // sum of accumulated values by 32 bits
134 accum = accum >> 23;
135 inactiveResult.fixed = accum;
136 inactiveResult.fixed |= signExp;
137 }
138
139 // calculate result inactive difference as percentage of full result
140 deltaResult = result - (double)inactiveResult.ieee;
141 deltaResult = fabs(deltaResult);
142 deltaResult = deltaResult/result;
143
144 // if over threshold bump result truncation count
145 if (deltaResult >= threshold) { multCount++; }
146
147 // truncate after normalization
148 inactiveResult.fixed &= normMask;
149
150 // calculate truncated inactive difference as percentage of full result
151 deltaResult = result - (double)inactiveResult.ieee;
152 deltaResult = fabs(deltaResult);
153 deltaResult = deltaResult / result;
154
155 // if over threshold bump result truncation count
156 if (deltaResult >= threshold) { normCount++; }
157
158 }
159 // copy value out to GPU memory
160 mult_result[index] = multCount;
161 norm_result[index] = normCount;
162 }
163
164 int main(int argc, char** argv) {
165
166     printf("threshold = %f\nmultiplier inactive mantissa bits = %d\ninactive mantissa\n",
167           threshold, inactiveBits, activeBits);
168     fflush(stdout);
169
170     printf("Allocating host memory\n");
171     fflush(stdout);
172
173     time_t startTime = time(NULL);
174
175     // allocate partial count array for output on host
176     uint32_t* host_mult_result = (uint32_t *)malloc(NUM_BLOCKS * NUM_THREADS *
177           sizeof(uint32_t));
178     if (host_mult_result == NULL) {
179         fprintf(stderr, "Failed to allocate host array!\n");
180         exit(EXIT_FAILURE);
181     }
182     uint32_t* host_norm_result = (uint32_t*)malloc(NUM_BLOCKS * NUM_THREADS *
183           sizeof(uint32_t));
184     if (host_norm_result == NULL) {
185         fprintf(stderr, "Failed to allocate host array!\n");
186         exit(EXIT_FAILURE);
187     }
188
189     printf("Allocating gpu memory\n");
190     fflush(stdout);
191
192     // Error code to check return values for CUDA calls
193     cudaError_t err = cudaSuccess;
194
195     // allocate partial count array for output on GPU
196     uint32_t* gpu_mult_result;
197     err = cudaMalloc((void **)&gpu_mult_result, NUM_BLOCKS * NUM_THREADS *

```

```

196   if (err != cudaSuccess) {
197       fprintf(stderr, "Failed to allocate gpu array!\n");
198       exit(EXIT_FAILURE);
199   }
200   uint32_t* gpu_norm_result;
201   err = cudaMalloc((void**)&gpu_norm_result, NUM_BLOCKS * NUM_THREADS *
202   sizeof(uint32_t));
203   if (err != cudaSuccess) {
204       fprintf(stderr, "Failed to allocate gpu array!\n");
205       exit(EXIT_FAILURE);
206   }
207
208   printf("Fire off GPU kernel\n");
209   fflush(stdout);
210
211   // fire off the CUDA kernel
212   fpMult << <NUM_BLOCKS, NUM_THREADS >> > (gpu_mult_result, gpu_norm_result);
213   err = cudaGetLastError();
214   if (err != cudaSuccess)
215   {
216       fprintf(stderr, "Failed to launch vectorAdd kernel (error code %s)!\n",
217       cudaGetErrorString(err));
218       exit(EXIT_FAILURE);
219   }
220   printf("Copy results from GPU to host\n");
221   fflush(stdout);
222
223   // Copy the device result vector in device memory to the host result vector
224   // in host memory.
225   err = cudaMemcpy(host_mult_result, gpu_mult_result, NUM_BLOCKS * NUM_THREADS *
226   sizeof(uint32_t), cudaMemcpyDeviceToHost);
227   if (err != cudaSuccess)
228   {
229       fprintf(stderr, "Failed to copy vector from device to host (error code %s)!\n",
230       cudaGetErrorString(err));
231       exit(EXIT_FAILURE);
232   }
233   err = cudaMemcpy(host_norm_result, gpu_norm_result, NUM_BLOCKS * NUM_THREADS *
234   sizeof(uint32_t), cudaMemcpyDeviceToHost);
235   if (err != cudaSuccess)
236   {
237       fprintf(stderr, "Failed to copy vector from device to host (error code %s)!\n",
238       cudaGetErrorString(err));
239       exit(EXIT_FAILURE);
240   }
241
242   uint64_t multCount = 0LL;
243   uint64_t normCount = 0LL;
244
245   printf("Sum up results\n");
246   fflush(stdout);
247
248   // sum up to create total count
249   for (int i = 0; i < (NUM_BLOCKS * NUM_THREADS); i++) {
250       multCount += host_mult_result[i];
251       normCount += host_norm_result[i];
252   }
253
254   double numPasses = ((double)NUM_BLOCKS) * ((double)NUM_THREADS) * (double)(1LL <<
255   23);
256
257   printf("%lld seconds\n", time(NULL)-startTime);

```

```
254     printf("multCount after truncate = %lld\n",
255           multCount,
256           (uint64_t)numPasses,
257           ((double)multCount / numPasses)*100.0);
258
259     printf("truncation after normalize %lld/%lld = %f%%\n",
260           normCount,
261           (uint64_t)numPasses,
262           ((double)normCount / numPasses) * 100.0);
263
264     // free GPU memory
265     err = cudaFree(gpu_mult_result);
266     if (err != cudaSuccess)
267     {
268         fprintf(stderr, "Failed to free device operand vector (error code %s)!\n",
269               cudaGetErrorString(err));
270         exit(EXIT_FAILURE);
271     }
272     err = cudaFree(gpu_norm_result);
273     if (err != cudaSuccess)
274     {
275         fprintf(stderr, "Failed to free device operand vector (error code %s)!\n",
276               cudaGetErrorString(err));
277         exit(EXIT_FAILURE);
278     }
279
280     // Free host memory
281     free(host_mult_result);
282     free(host_norm_result);
283 }
```

ATTACHMENT B2

```

1  #include <stdint.h>
2  #include <stdint.h>
3  #include <stdlib.h>
4  #include <math.h>
5  #include <time.h>
6  #include <cuda_runtime.h>
7  #include <helper_cuda.h>
8
9  // number of active bits
10 // #define activeBits (10)
11 // error threshold
12 // #define threshold (0.0005)
13
14 // number of inactive bits
15 #define inactiveBits (37)
16 #define activeBits (46 - inactiveBits)
17 // error threshold
18 #define threshold (0.0005)
19
20 // CUDA parameters
21 #define NUM_THREADS 1024
22 #define NUM_BLOCKS 8192
23
24 // mask out the inactive LSB from the accumulator
25 #define inactiveMask (0xffffffffffffffffLL<<inactiveBits)
26 // mask out the inactive LSB from the result after normalization
27 #define normMask (0xffffffff<<(inactiveBits-23))
28
29 // ieee754 1.0
30 #define signExp (0x3f800000)
31 // lsb of exponent in ieee754
32 #define firstExpBit (0x800000)
33
34 // access value as int32 or float
35 union rawFloat {
36     uint32_t fixed;
37     float ieee;
38 };
39
40 __global__ void
41 fpMult(uint32_t *mult_result, uint32_t *norm_result) {
42
43     // operand mantissas
44     uint32_t mantissa1;
45     uint32_t mantissa2;
46
47     // full ieee754 operands
48     union rawFloat operand1;
49     union rawFloat operand2;
50
51     // single ieee754 result
52     float result;
53
54     // ieee754 result with inactive bits
55     union rawFloat inactiveResult;
56
57     // truncated result difference
58     double deltaResult;
59
60     // index into operand2 bits
61     int bit;
62
63     // 64 bit shift and add accumulator
64     uint64_t accum;
65
66     // operand mantissas with implicit 1. added

```



```

67  uint32_t mult1;
68  uint32_t mult2;
69
70  // count of error over threshold for multiplier truncation
71  uint32_t multCount;
72
73  // count of error over threshold for truncation after normalization
74  uint32_t normCount;
75
76  // index into count array
77  uint32_t index;
78
79  // cycle through all mantissas for operand 1
80  index = (blockDim.x * blockIdx.x) + threadIdx.x;
81  mantissa1 = index;
82
83  // build the first operand
84  operand1.fixed = signExp | mantissa1;
85
86  multCount = 0;
87  normCount = 0;
88
89  // all ieee floating point values for second operand >= 1.0 and < 2.0
90  for(mantissa2 = 0; mantissa2 < (1<<23); mantissa2++) {
91
92      // build the second operand
93      operand2.fixed = signExp | mantissa2;
94
95      // reference ieee754 single precision result
96      result = operand1.ieee * operand2.ieee;
97
98      // extract the mantissa
99      mult1 = mantissa1;
100     mult2 = mantissa2;
101
102     // set the implicit 1.
103     mult1 |= firstExpBit;
104     mult2 |= firstExpBit;
105
106     //
107     // multiply implemented as shift and add
108     //
109     accum = 0LL;
110
111     // iterate over 24 bit mantissa value
112     for(bit=0;bit<24;bit++) {
113
114         // for each bit position in mantissa1 add shifted mantissa2
115         if((mult1>>bit)&1) {
116             accum += (((uint64_t)mult2) << bit) & inactiveMask;
117         }
118     }
119
120     // normalize result
121     if(accum >> 47) {
122
123         // shift accumulated value by 24 bits
124         accum = accum >> 24;
125         inactiveResult.fixed = accum;
126         inactiveResult.fixed |= signExp;
127
128         // increment the exponent
129         inactiveResult.fixed += firstExpBit;
130
131     } else {
132

```

```

133 // shift accumulated value by 32 bits
134 accum = accum >> 23;
135 inactiveResult.fixed = accum;
136 inactiveResult.fixed |= signExp;
137 }
138
139 // calculate result inactive difference as percentage of full result
140 deltaResult = (double)result - (double)inactiveResult.ieee;
141 deltaResult = fabs(deltaResult);
142 deltaResult = deltaResult/result;
143
144 // if over threshold bump result truncation count
145 if (deltaResult >= threshold) { multCount++; }
146
147 // truncate after normalization
148 inactiveResult.fixed &= normMask;
149
150 // calculate truncated inactive difference as percentage of full result
151 deltaResult = (double)result - (double)inactiveResult.ieee;
152 deltaResult = fabs(deltaResult);
153 deltaResult = deltaResult / result;
154
155 // if over threshold bump result truncation count
156 if (deltaResult >= threshold) { normCount++; }
157
158 }
159 // copy value out to GPU memory
160 mult_result[index] = multCount;
161 norm_result[index] = normCount;
162 }
163
164 int main(int argc, char** argv) {
165
166     printf("threshold = %f\nmultiplier inactive mantissa bits = %d\ninactive mantissa\n",
167           threshold, inactiveBits, activeBits);
168     fflush(stdout);
169
170     printf("Allocating host memory\n");
171     fflush(stdout);
172
173     time_t startTime = time(NULL);
174
175     // allocate partial count array for output on host
176     uint32_t* host_mult_result = (uint32_t *)malloc(NUM_BLOCKS * NUM_THREADS *
177           sizeof(uint32_t));
178     if (host_mult_result == NULL) {
179         fprintf(stderr, "Failed to allocate host array!\n");
180         exit(EXIT_FAILURE);
181     }
182     uint32_t* host_norm_result = (uint32_t*)malloc(NUM_BLOCKS * NUM_THREADS *
183           sizeof(uint32_t));
184     if (host_norm_result == NULL) {
185         fprintf(stderr, "Failed to allocate host array!\n");
186         exit(EXIT_FAILURE);
187     }
188
189     printf("Allocating gpu memory\n");
190     fflush(stdout);
191
192     // Error code to check return values for CUDA calls
193     cudaError_t err = cudaSuccess;
194
195     // allocate partial count array for output on GPU
196     uint32_t* gpu_mult_result;
197     err = cudaMalloc((void **)&gpu_mult_result, NUM_BLOCKS * NUM_THREADS *

```

```

196   if (err != cudaSuccess) {
197       fprintf(stderr, "Failed to allocate gpu array!\n");
198       exit(EXIT_FAILURE);
199   }
200   uint32_t* gpu_norm_result;
201   err = cudaMalloc((void**)&gpu_norm_result, NUM_BLOCKS * NUM_THREADS *
202   sizeof(uint32_t));
203   if (err != cudaSuccess) {
204       fprintf(stderr, "Failed to allocate gpu array!\n");
205       exit(EXIT_FAILURE);
206   }
207
208   printf("Fire off GPU kernel\n");
209   fflush(stdout);
210
211   // fire off the CUDA kernel
212   fpMult << <NUM_BLOCKS, NUM_THREADS >> > (gpu_mult_result, gpu_norm_result);
213   err = cudaGetLastError();
214   if (err != cudaSuccess)
215   {
216       fprintf(stderr, "Failed to launch vectorAdd kernel (error code %s)!\n",
217       cudaGetErrorString(err));
218       exit(EXIT_FAILURE);
219   }
220   printf("Copy results from GPU to host\n");
221   fflush(stdout);
222
223   // Copy the device result vector in device memory to the host result vector
224   // in host memory.
225   err = cudaMemcpy(host_mult_result, gpu_mult_result, NUM_BLOCKS * NUM_THREADS *
226   sizeof(uint32_t), cudaMemcpyDeviceToHost);
227   if (err != cudaSuccess)
228   {
229       fprintf(stderr, "Failed to copy vector from device to host (error code %s)!\n",
230       cudaGetErrorString(err));
231       exit(EXIT_FAILURE);
232   }
233   err = cudaMemcpy(host_norm_result, gpu_norm_result, NUM_BLOCKS * NUM_THREADS *
234   sizeof(uint32_t), cudaMemcpyDeviceToHost);
235   if (err != cudaSuccess)
236   {
237       fprintf(stderr, "Failed to copy vector from device to host (error code %s)!\n",
238       cudaGetErrorString(err));
239       exit(EXIT_FAILURE);
240   }
241
242   uint64_t multCount = 0LL;
243   uint64_t normCount = 0LL;
244
245   printf("Sum up results\n");
246   fflush(stdout);
247
248   // sum up to create total count
249   for (int i = 0; i < (NUM_BLOCKS * NUM_THREADS); i++) {
250       multCount += host_mult_result[i];
251       normCount += host_norm_result[i];
252   }
253
254   double numPasses = ((double)NUM_BLOCKS) * ((double)NUM_THREADS) * (double)(1LL <<
255   23);
256
257   printf("%lld seconds\n", time(NULL)-startTime);

```

```
254 printf("multCount after truncation = %lld\n",
255        multCount,
256        (uint64_t)numPasses,
257        ((double)multCount / numPasses)*100.0);
258
259 printf("truncation after normalize %lld/%lld = %f%%\n",
260        normCount,
261        (uint64_t)numPasses,
262        ((double)normCount / numPasses) * 100.0);
263
264 // free GPU memory
265 err = cudaFree(gpu_mult_result);
266 if (err != cudaSuccess)
267 {
268     fprintf(stderr, "Failed to free device operand vector (error code %s)!\n",
269             cudaGetErrorString(err));
270     exit(EXIT_FAILURE);
271 }
272 err = cudaFree(gpu_norm_result);
273 if (err != cudaSuccess)
274 {
275     fprintf(stderr, "Failed to free device operand vector (error code %s)!\n",
276             cudaGetErrorString(err));
277     exit(EXIT_FAILURE);
278 }
279
280 // Free host memory
281 free(host_mult_result);
282 free(host_norm_result);
283 }
```

ATTACHMENT C1

```
1  #include <stdio.h>
2
3  int main(int argv, char **argc) {
4      int v1 = 0;
5      int v2 = 0;
6
7      int u1 = 0;
8      int u2 = 0;
9
10     int pairs = 0;
11
12     int opr1, opr2;
13     for(opr1 = -126; opr1 < 128; opr1++) {
14         for(opr2 = -126; opr2 < 128; opr2++) {
15             if(opr1+opr2 > 127) v1++;
16             if(opr1+opr2 == 127) v2++;
17
18             if(opr1+opr2 < -127) u1++;
19             if(opr1+opr2 == -127) u2++;
20
21             pairs++;
22         }
23     }
24     printf("pairs = %d, v1 = %d, v2 = %d u1 = %d u2 = %d\n",
25           pairs, v1, v2, u1, u2);
26 }
27
```